WebSphere MQ for Linux (Intel) V5.3 - Performance Evaluations

Version 1.0

24th October 2002

Mark Orchard
Peter Toghill.
WebSphere MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

Take Note!

Before using this report, be sure to read the general information under “Notices”.

First Edition, November 2002
This edition applies to V1.0 of WebSphere MQ for Linux V5.3 – Performance Evaluations and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This report is intended to help the reader understand the performance characteristics of WebSphere MQ for Linux V5.3. The information is not intended as the specification of any programming interfaces that are provided by WebSphere MQ.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which it operates.

Information contained in this report has **not** been submitted to any formal IBM test and is distributed "**as-is**". The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the reader to evaluate the information and project the results to their operational environment.

The performance measurements included in this report were measured in a controlled environment and the results obtained in other environments may vary significantly.

Trademarks and service marks:

The following terms used in this publication are trademarks of the IBM Corporation in the United States or other countries or both:

IBM

MQSeries

WebSphere MQ

SupportPac

FFST

Red Hat is a registered trademark of Red Hat, Inc

Intel and xeon are registered trademarks of Intel Corporation

Linux is a registered trademark of Linus Torvalds

Preface

Target audience

This SupportPac is designed for people who:

- Will be designing and implementing solutions using WebSphere MQ for Linux
- Want to understand the performance limits of WebSphere MQ for Linux V5.3
- Want to understand what actions may be taken to tune WebSphere MQ for Linux

The reader should have a general awareness of the Linux Red Hat Operating System and of MQSeries in order to make best use of this SupportPac. Readers should read the section '**How this document is arranged**'—**Page iii** to familiarise themselves with where specific information can be found for later reference.

The contents of this SupportPac

This SupportPac includes:

- Release highlights performance charts,
- Performance measurements with figures and tables to present the performance capabilities of WebSphere MQ local queue manager, client channel, and distributed queuing scenarios,
- Interpretation of the results and implications on designing or sizing WebSphere MQ local queue manager, client channel, and distributed queuing configurations.

Feedback on this SupportPac

We welcome constructive feedback on this report. Does it provide the sort of information you want? Do you feel something important is missing? Is there too much technical detail, or not enough? Could the material be presented in a manner more useful to you? Please direct any comments of this nature to: **WMQPG@uk.ibm.com**.

Specific queries about performance problems on your WebSphere MQ system should be directed to your local IBM Representative or Support Center.

Acknowledgements

The author is very grateful to Richard Eures for help in producing this report.

Introduction

The three scenarios in this report used to generate the performance data are classified into: the local queue manager scenario, the client channel scenario, and the distributed queuing scenario. The performance improvements in WebSphere MQ V5.3 can be divided into two areas:

- queue manager enhancements, and
- channel capacity enhancements.

The enhancements to the queue manager are apparent through many of the measurements in this report where WebSphere MQ V5.3 is compared to Version 5.2. Channel capacity enhancements are covered briefly in the *release highlights* section and in more detail towards the end of the report.

Unless otherwise specified, the standard message sized used for all the measurements in this report is 2K (2,048 bytes), trusted channels using the inetd 'amqcrsta' listener, and nontrusted threaded server application are used.

A Netfinity 4-way Intel (xeon) 700MHz with 8GB of RAM was used as the device under test for all the measurements in this report.

How this document is arranged

Release highlights

Pages: 1-3

Section one outlines the major performance *improvements* achieved in WebSphere MQ V5.3 compared to Version 5.2. The highlights are a subset of the results shown in the performance *headlines* section.

Performance headlines

Pages: 4-15

Section two of the document contains the performance *headlines* for each of the three test scenarios, with MQI applications connected to:

- a local queue manager,
- to a remote queue manager over MQI-client channels, and
- to a local queue manager, driving throughput between the local and remote queue manager, over server channel pairs.

The headline tests show:

- the *maximum* message throughput achieved with an increasing number of MQI applications,
- the *maximum* message throughput achieved using MQI-clients connected to a queue manager
- the *maximum* message throughput achieved over server channel pairs between two queue managers.

Large messages

Pages: 17-26

Section three of the document contains performance measurements for *large messages*. This includes *MQI response times* of 50byte to 2MB messages, and *20K* and *200K* messages using the same test scenarios as for the performance *headlines*.

CONTENTS

TABLES

FIGURES

1 Release highlights

1.1 Improvements to nonpersistent and persistent messaging

- Nonpersistent Messages
  - 70%: In a local queue manager environment
  - 47%: In an MQI-client environment,
  - 40%: In a distributed queuing environment.

- Persistent messages
  - 216%: In a local queue manager environment
  - 198%: In an MQI-client environment
  - 102%: In a distributed queuing environment.

1.2 Peak message throughput – local queue manager

Figure 1 shows the peak round trips per second achieved for nonpersistent and persistent messages with a local queue manager, MQSeries V5.2 vs. WebSphere MQ V5.3.

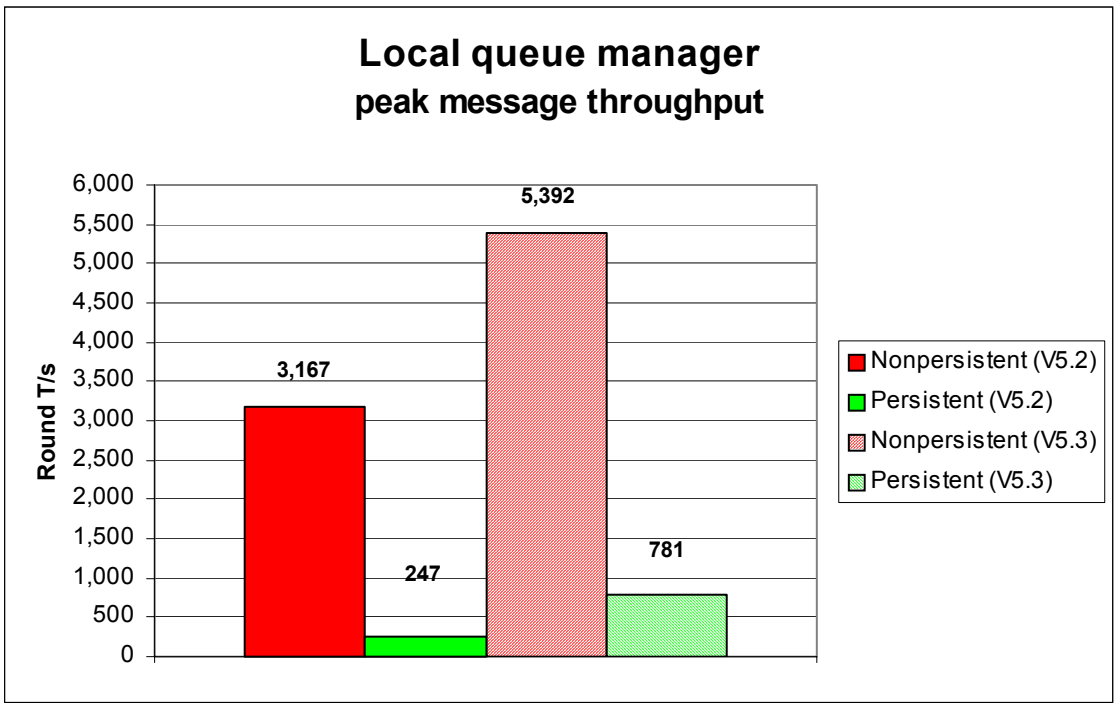

Figure 1 – Peak message throughput, local queue manager

Note: messaging in these tests is with no think-time.

1.3 Peak message throughput – client channels

Figure 2 below shows the peak round trips per second achieved for nonpersistent and persistent messages with MQI-client channels, MQSeries V5.2 vs. WebSphere MQ V5.3.

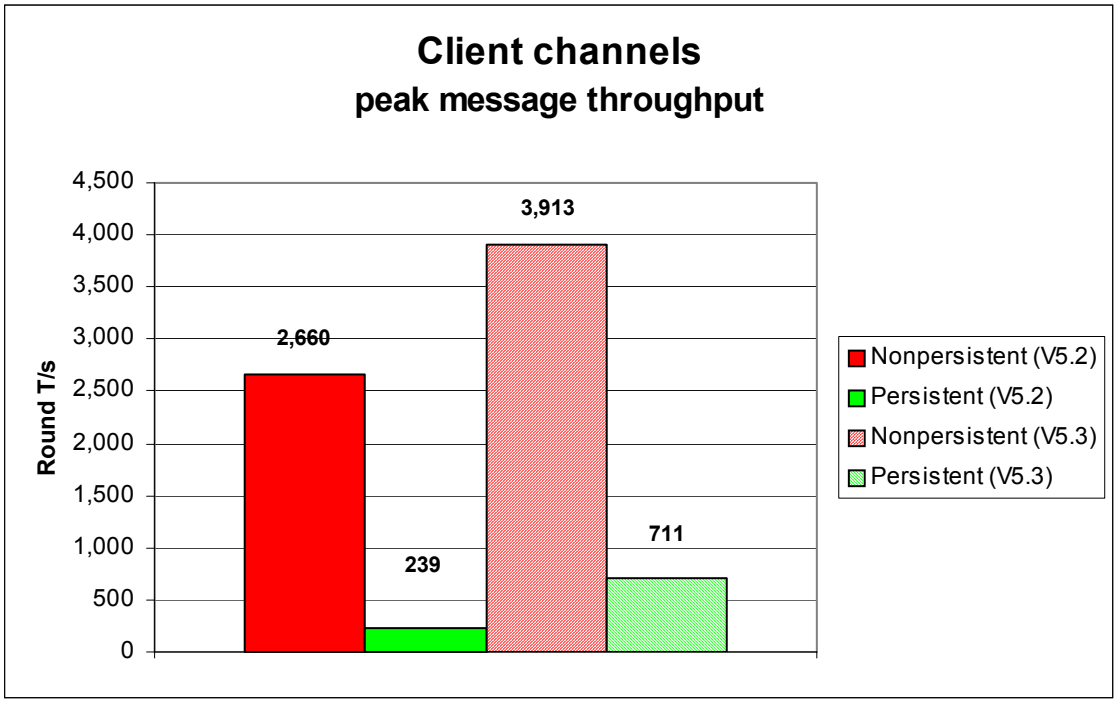

Figure 2 – Peak message throughput, client channels

Note: messaging in these tests is with no think-time.

1.4 Peak message throughput – distributed queuing

Figure 3 shows the peak round trips per second achieved for nonpersistent and persistent messages with server channels, MQSeries V5.2 vs. WebSphere MQ for V5.3.

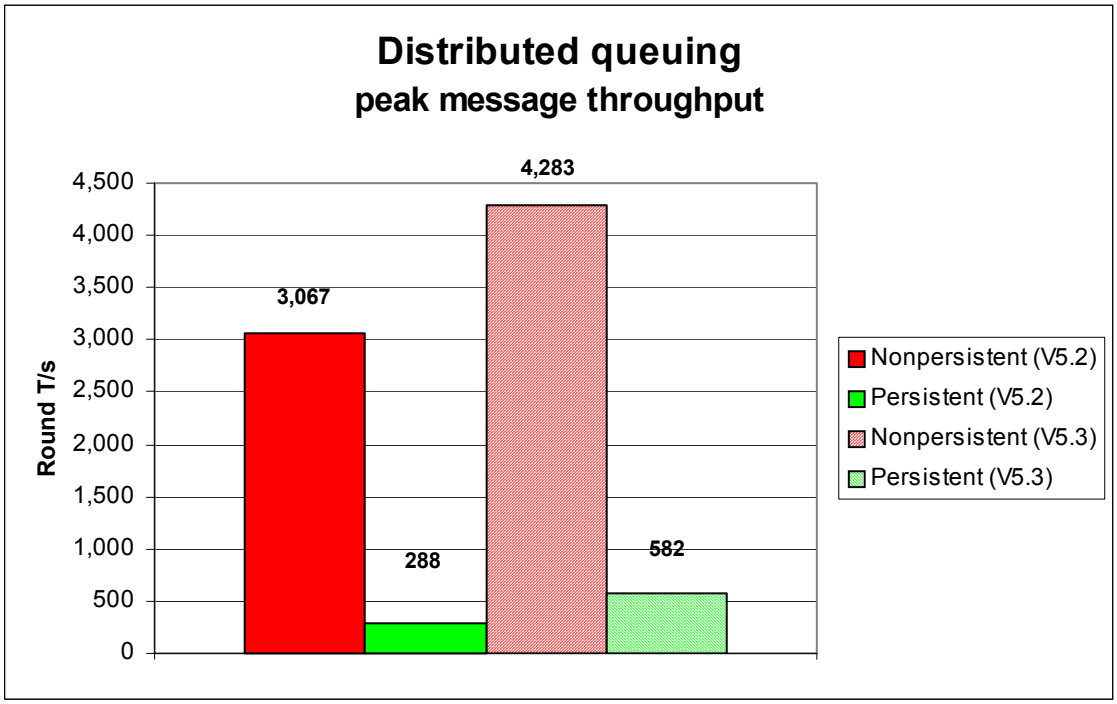

Figure 3 – Peak message throughput, distributed queuing

Note: messaging in these tests is with no think-time.

2 Performance headlines

The measurements for the local queue manager scenario are for processing messages with no *think-time*. For the client channel scenario and distributed queuing scenario, there are also measurements for *rated* messaging.

No think-time is defined as when the driving applications do not wait after getting a reply message before submitting subsequent request messages—this is also referred to as *tight-loop*.

In the rated messaging tests, the rate used is 1 round trip per driving application per *second*. In the client channel test scenarios, each driving application using a dedicated MQI-client channel, in the distributed queuing test scenarios, one or more applications submit messages over a fixed number of server channels.

All tests are automatically stopped after the response time of 1 round trip exceeds 1 second.

2.1 Local queue manager test scenario

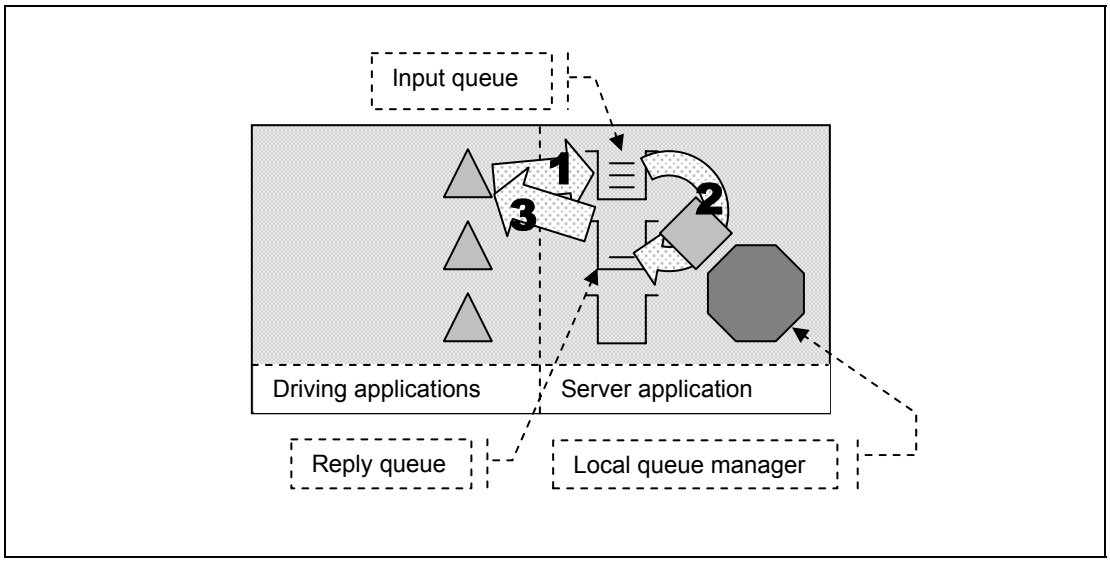

Figure 4 – Connections into a local queue manager

1) The driving application puts a message to the common input queue on the local queue manager, and holds on to the message identifier returned in the message descriptor. The driving application then waits indefinitely for a reply to arrive on the common reply queue.

2) The server application gets messages from the common input queue and places a reply to the common reply queue. The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.

3) The driving application gets a reply from the common reply queue using the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the local queue manager tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in '**MQI response times: 50bytes to 2MB – local queue manager**'—**Page 17**, and '**Large messages: 20K and 200K – local queue manager**'—**Page 20**

2.1.1 Nonpersistent messages – local queue manager

Figure 5 and **Table 1** show that the peak throughput of nonpersistent messages has increased in Version 5.3 compared to Version 5.2. Using 8 driving applications nonpersistent throughput is improved by 78% (3,021 cf. 5,392 RT/s). Version 5.3 shows a consistent throughput improvement — which is important in most queue manager systems.

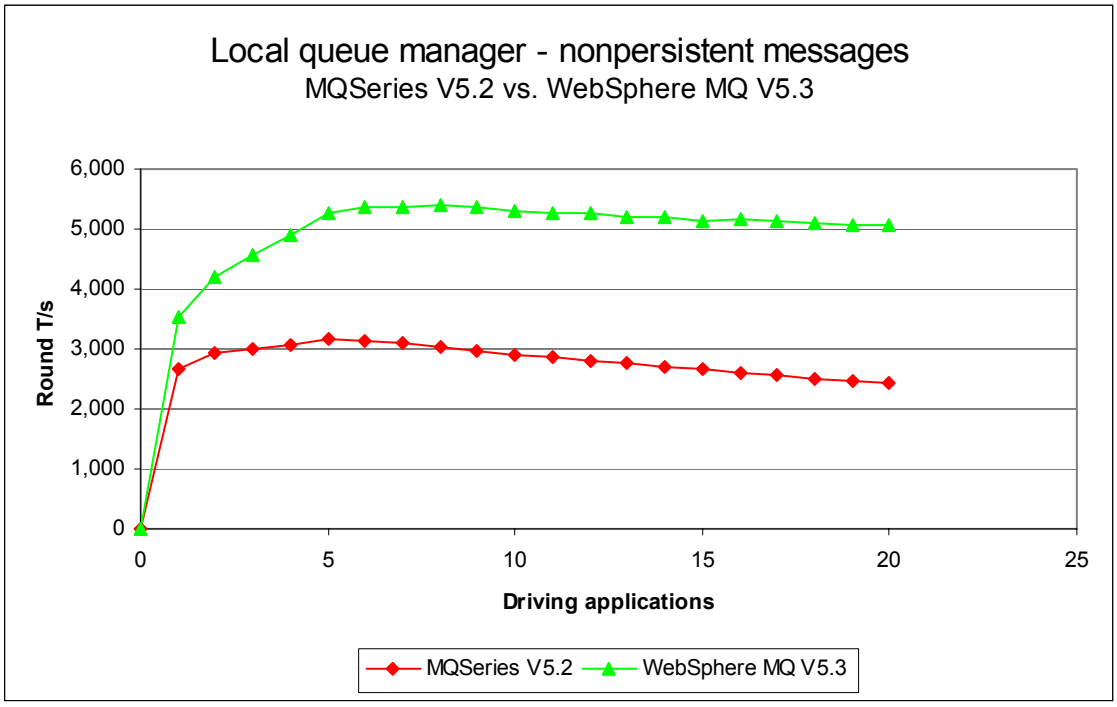

Figure 5 – Performance headline, nonpersistent messages, and local queue manager

Note: messaging in these tests is with no think-time.

Test name	Product version	Apps	Round T/s	%	Resp time (s)
local_np1	MQSeries V5.2	**5** (8) (20)	**3,167** (3,021) (2,422)	n/a	0.002 (0.003) (0.010)
local_np1	WebSphere MQ V5.3	(5) **8** (20)	(5,251) **5,392** (5,077)	(+66) +78 (+110)	(0.001) 0.002 (0.005)

Table 1 – Performance headline, nonpersistent messages, local queue manager

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

2.1.2 Persistent messages – local queue manager

Figure 6 and **Table 2** below show that the peak throughput of persistent messages has increased by 216% (247 cf. 780 RT/s) comparing Version 5.2 to Version 5.3. Using 20 driving applications persistent throughput is improved by 52% (247 cf. 377 RT/s). Version 5.3 has the advantage of improved scalability when using 8 or more driving applications.

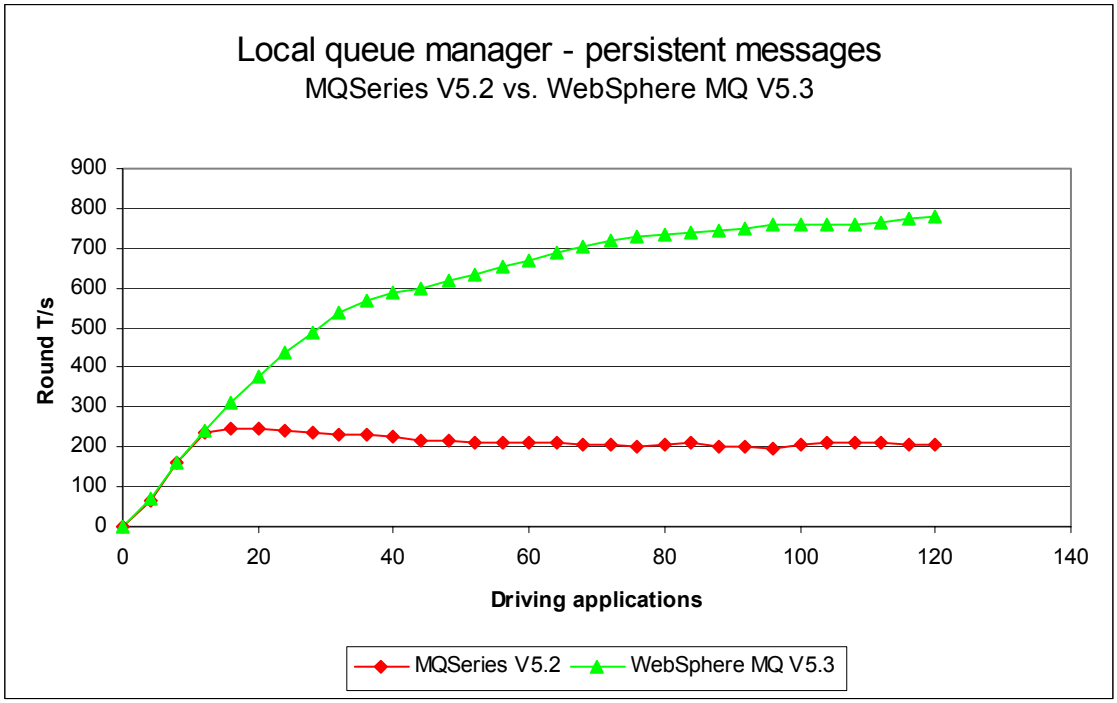

Figure 6 – Performance headline, persistent messages, and local queue manager

Note: messaging in these tests is with no think-time.

Test name	Product version	Apps	Round T/s	%	Resp time (s)
local_pm1	MQSeries V5.2	**20** (120)	**247** (208)	n/a	0.09 (0.638)
local_pm1	WebSphere MQ V5.3	(20) **120**	(377) **780**	(+52) +276	(0.063) 0.184

Table 2 – Performance headline, persistent messages, local queue manager

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

2.2 Client channels test scenario

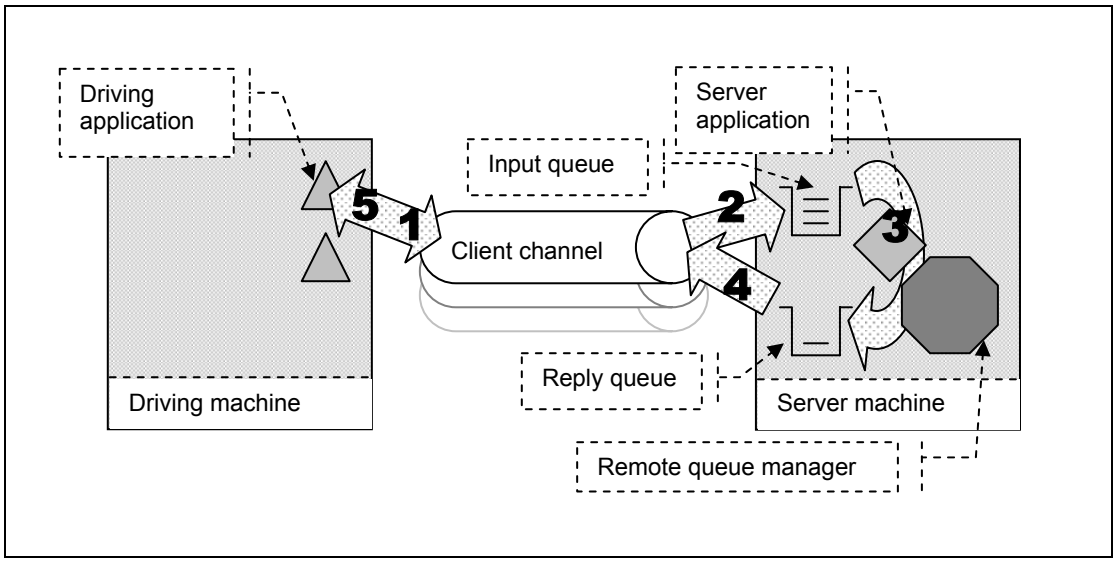

Figure 7 – MQI-client channels into a remote queue manager

1, 2) The driving application puts a request message (over a client channel), to the common input queue, and holds on to the message identifier returned in the message descriptor. The driving application then waits indefinitely for a reply to arrive on the common reply queue.

3) The server application gets messages from the common input queue and places a reply to the common reply queue. The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.

4, 5) The driving application gets the reply message (over the client channel), from the common reply queue. The driving application uses the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the client channel tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in '**Large messages: 20K and 200K – client channels**'—**Page 23**.

2.2.1 Nonpersistent Messages – Client channels

Figure 8 and **Table 3** below show that the peak throughput of nonpersistent messages is higher comparing Version 5.2 to Version 5.3, with the advantage of improved scalability when using as few as 5 driving applications (throughput up by 47% : 2,660 cf. 3,913 RT/s). Using 20 driving applications throughput is improved by 75% : 2,153 cf. 3,765 RT/s).

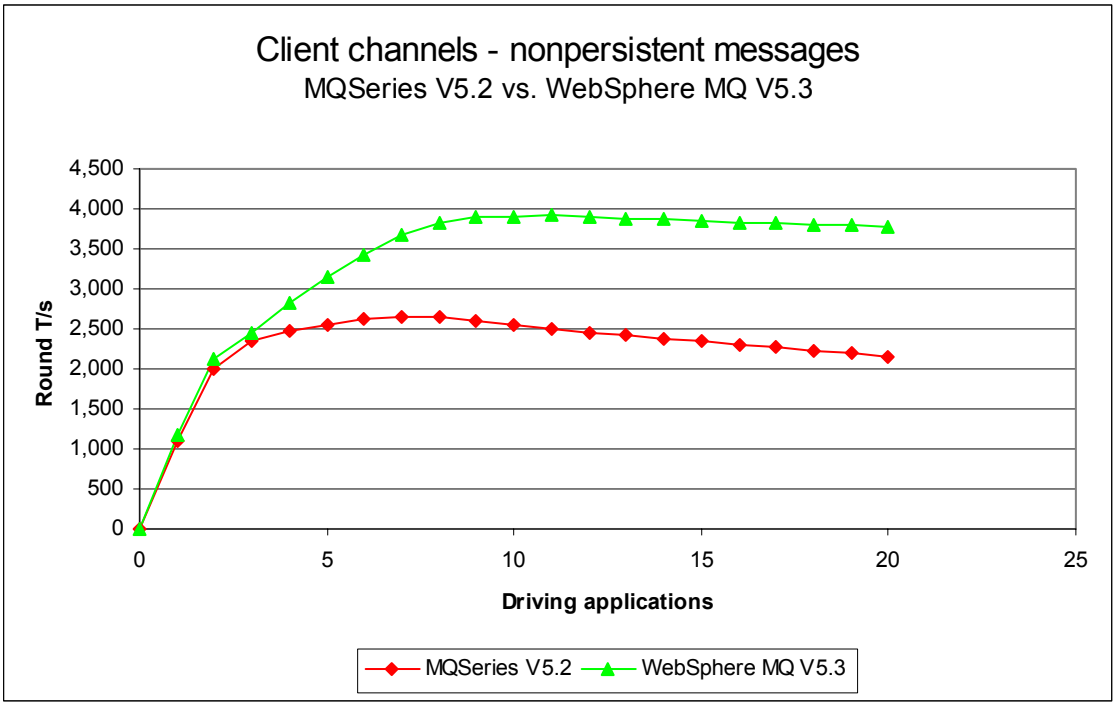

Figure 8 – Performance headline, nonpersistent messages, and client channels

Note: messaging in these tests is with no think-time.

Test name	Product version	Apps	Round T/s	%	Resp time (s)
clnp1 (inetd)	MQSeries V5.2	**7** (11) (20)	**2,660** (2,507) (2,153)	n/a	0.003 (0.005) (0.011)
clnp1 (inetd)	WebSphere MQ V5.3	(7) **11** (20)	(3,663) **3,913** (3,765)	(+38) +56 (+75)	(0.002) 0.003 (0.012)

Table 3 – Performance headline, nonpersistent messages, and client channels

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

2.2.2 Persistent messages – client channels

Figure 9 and **Table 4** below show that the peak throughput of persistent messages has increased by 200% (239 cf. 717 Round T/s) comparing Version 5.2 to Version 5.3, with the advantage of improved scalability giving the most performance improvement using one hundred or more driving applications (throughput up by 266% : 194 cf. 717 RT/s at 120 apps).

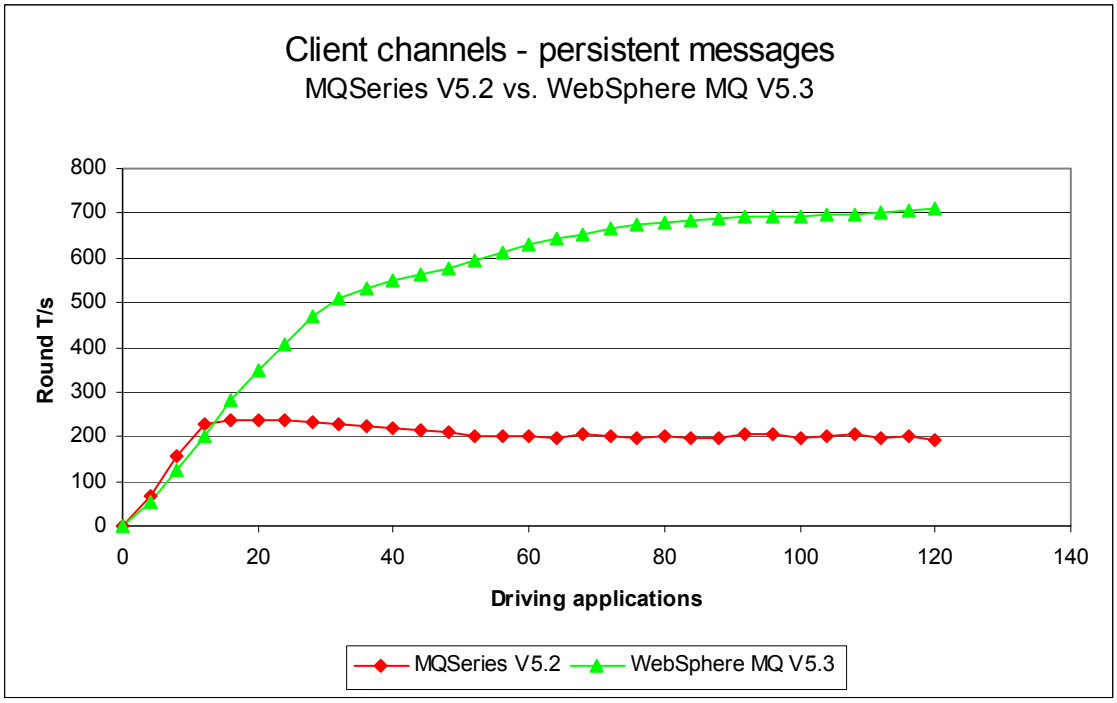

Figure 9 – Performance headline, persistent messages, and client channels

Note: messaging in these tests is with no think-time.

Test name	Product version	Apps	Round T/s	%	Resp time (s)
clpm3 (inetd)	MQSeries V5.2	**16** (120)	**239** (194)	n/a	0.076 (0.781)
clpm3 (inetd)	WebSphere MQ V5.3	(16) **120**	(280) **717**	(+17) **+266**	(0.124) **0.341**

Table 4– Performance headline, persistent messages, and client channels

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

2.2.3 'runmqlsr' vs. inetd 'amqcrsta' listener – client channels

For the following client channel measurements, the messaging rate used is 1 round trip per *second* per MQI-client channel, i.e. a request message outbound over the client channel and a reply message inbound over the channel per second. These tests also compare the difference between *nonthreaded* channels (the 'amqcrsta' process started by inetd) with *threaded* channels (the 'runmqlsr' process started by the user).

Figure 10 and Table 9 below show how the 'runmqlsr' and inetd 'amqcrsta' listeners in WebSphere MQ V5.3 give improved scalability by permitting a larger number of MQI-client connections into a single queue manager. In Version 5.3, it is now possible to connect more than 1,000 driving application into a single queue manager. Furthermore, the 'runmqlsr' has a reduced resource utilisation (one thread per connection compared to a process per connection for the 'amqcrsta' listener, a smaller memory footprint, less System V IPC), so is now the **preferred** method of running client channels and server channels.

Figure 10 below compares the swap reservation of WebSphere MQ V5.3 MQI-client connections for the inetd 'amqcrsta' listener and 'runmqlsr' listener.

An approximate formula for calculating swap reservation in MB using the inetd listener is

swap = 2 x No. driving apps

in addition, an approximate formula for calculating swap reservation in MB using the runmqlsr is

swap = 0.625 x No. driving apps

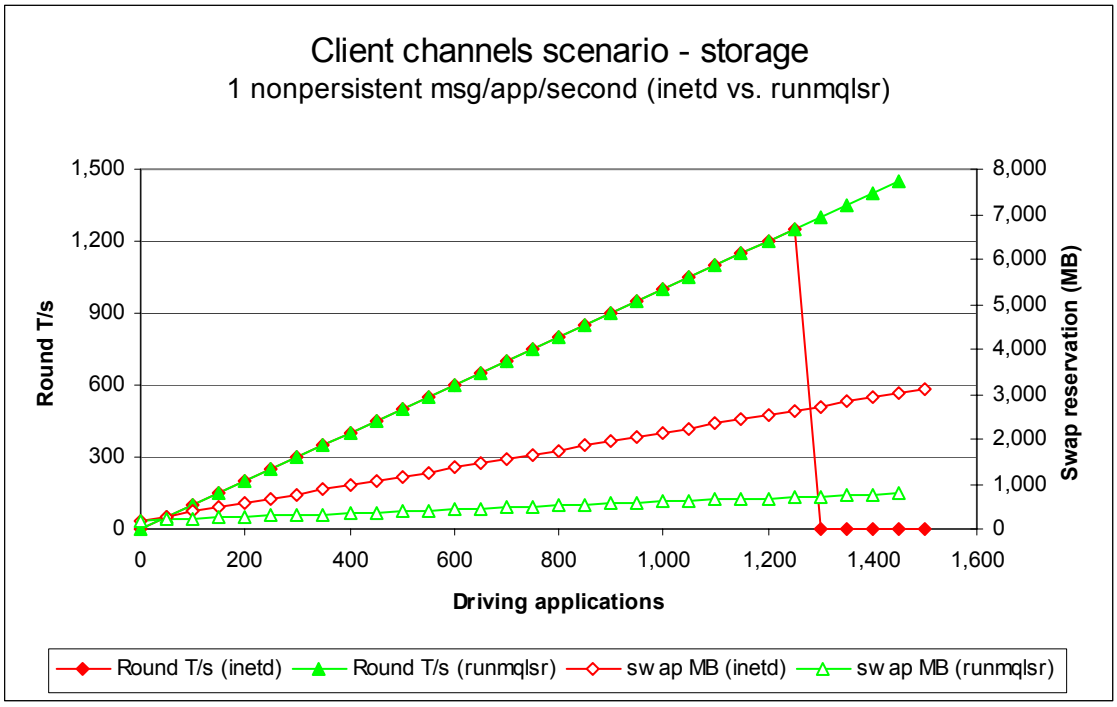

Figure 10 - 'runmqlsr' vs. inetd 'amqcrsta' listener, client channels

Note: messaging in these tests is 1 round trip per driving application per *second*.

Test name	Apps	Rate/App/hr	Round T/s	%	Resp time (s)
clnp1_r3600 (inetd) (MQSeries V5.2)	1,250 (950)	3,600	1,249 (949)	+31	0.006 (0.010)
clnp1_r3600_runmqlsr (MQSeries V5.2)	2,350 (950)	3,600	2,348 (950)	+147	0.008 (0.005)
clpm3_r3600 (inetd) (MQSeries V5.2)	700 (250)	3,600	699 (117)	+497	0.118 (0.062)
clpm3_r3600_runmqlsr (MQSeries V5.2)	700 (200)	3,600	699 (190)	+268	0.095 (0.202)

Table 5 – 1 round trip per driving application per *second*, client channels

Note: messaging in these tests is 1 round trip per driving application per *second*.

2.3 Distributed queuing test scenario

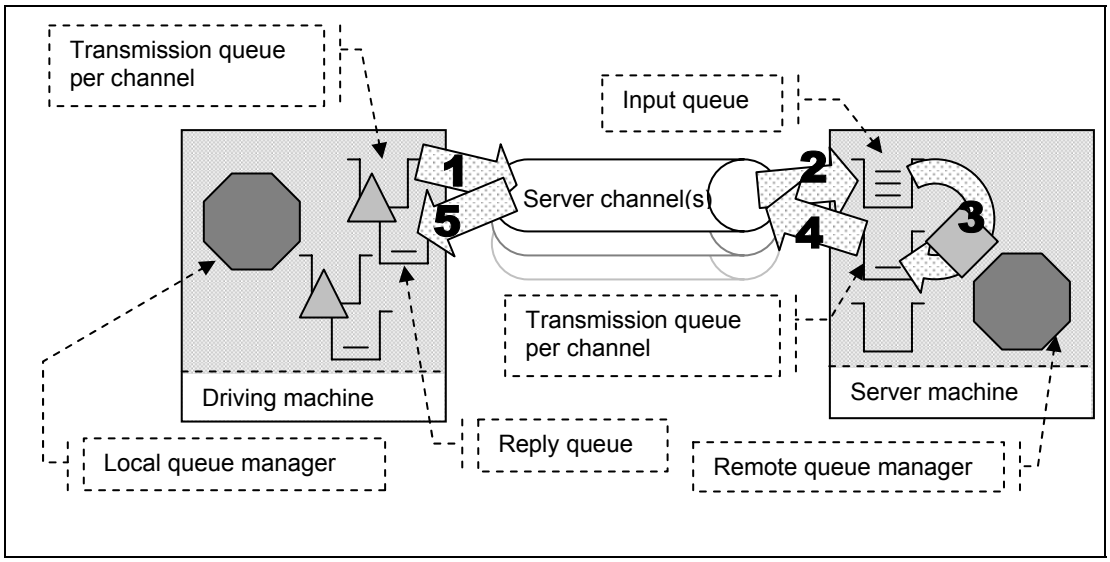

Figure 11 – Server channels between two queue managers

1) The driving application puts a message to a local definition of a remote queue located on the server machine, and holds on to the message identifier returned in the message descriptor. The driving application then waits indefinitely for a reply to arrive on a local queue.

2) The message channel agent takes messages off the channel and places them on the common input queue on the server machine.

3) The server application gets messages from the common input queue, and places a reply to the queue name extracted from the messages descriptor (the name of a local definition of a remote queue located on the driving machine). The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.

4) The message channel agent takes messages off the transmission queue and sends them over the channel to the driving machine.

5) The driving application gets a reply from a local queue. The driving application uses the message identifier held from when the request message was put to the local definition of the remote queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the distributed queuing tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in '**Large messages: 20K and 200K – distributed queuing**'—**Page 26**.

2.3.1 Nonpersistent messages – server Channels

Figure 12 and **Table 6** show that the peak throughput of nonpersistent messages has increased by 40% (3,067 cf. 4,283 RT/s) comparing Version 5.2 to Version 5.3, also with the advantage of improved scalability when using as few as 3 driving applications (throughput up by 8% : 2,364 cf. 2,545 RT/s).

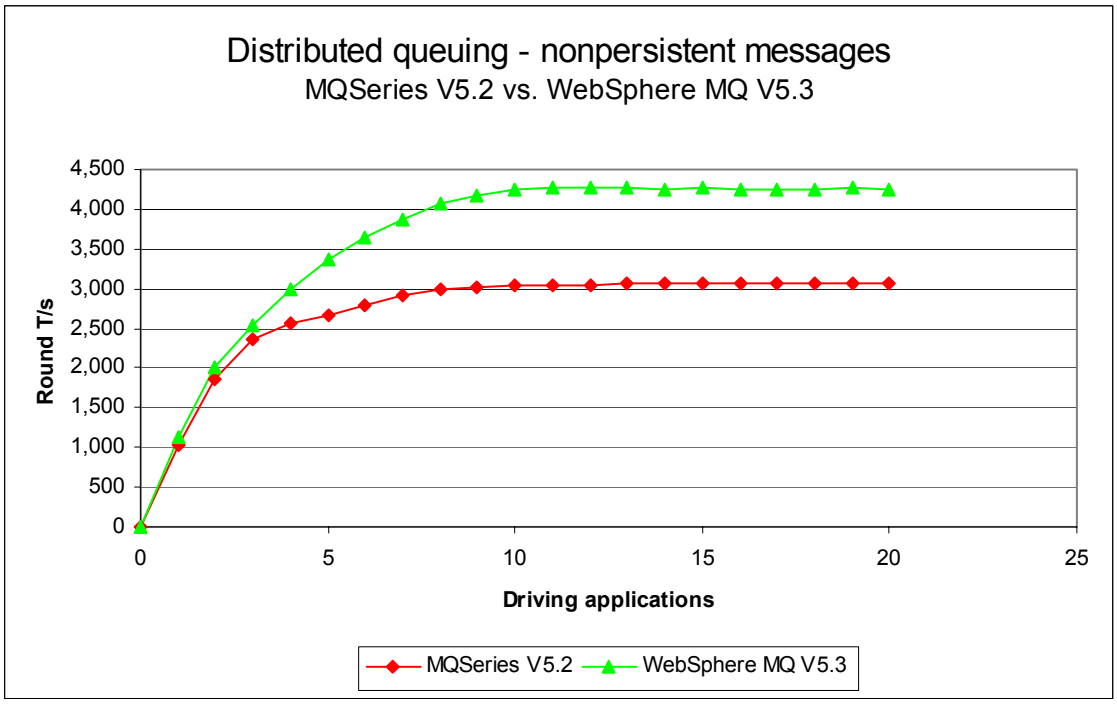

Figure 12 – Performance headline, nonpersistent messages, and server channels

Note: messaging in these tests is with no think-time.

Test name	Product version	Apps	Round T/s	%	Resp time (s)
dqnp1 (inetd)	MQSeries V5.2	**17** (13)	**3,067** (3,059)	n/a	0.005 (0.011)
dqnp1 (inetd)	WebSphere MQ V5.3	(17) **13**	(4,261) **4,283**	(+39) +40	(0.005) 0.003

Table 6 – Performance headline, nonpersistent messages, and server channels

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

2.3.2 Persistent messages – server channels

Figure 13 and **Table 7** below show that at 120 driving applications persistent message throughput has increased by 105% (284 cf. 582 RT/s) comparing Version 5.2 to Version 5.3, also with the advantage of improved scalability after 40 driving applications.

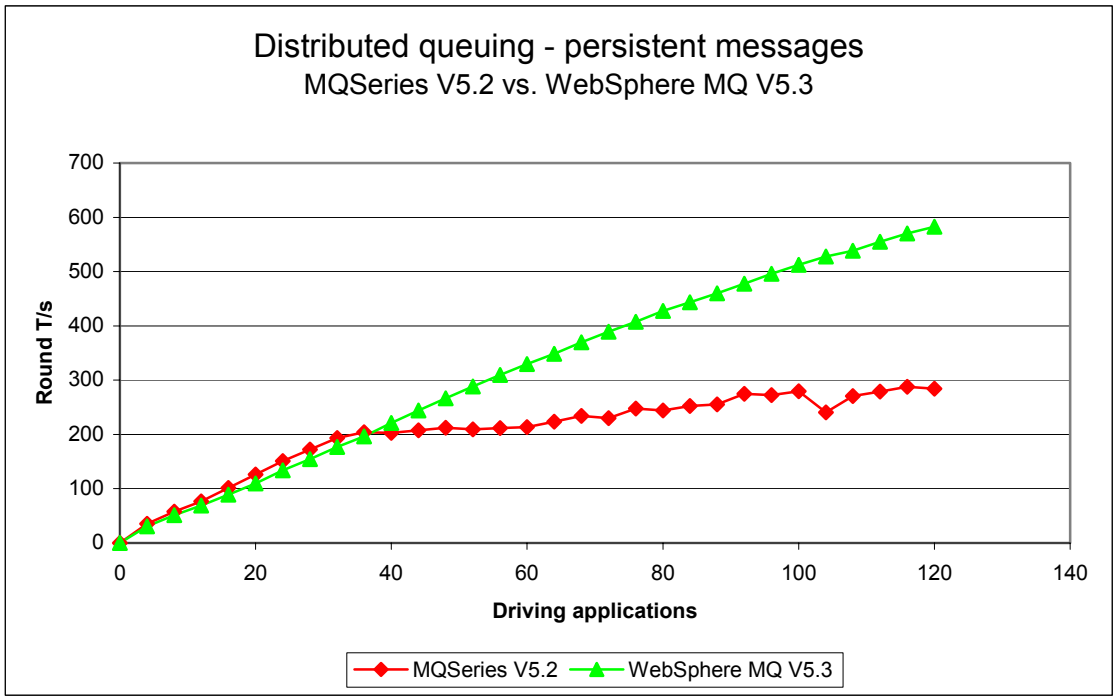

Figure 13 – Performance headline, persistent messages, and server channels

Note: messaging in these tests is with no think-time.

Test name	Product version	Apps	Round T/s	%	Resp time (s)
dqpm1 (inetd)	MQSeries V5.2	116 (120)	288 (284)	n/a	0.468 (0.533)
dqpm1 (inetd)	WebSphere MQ V5.3	(116) **120**	(570) **582**	(+98) **+105**	(0.249) **0.239**

Table 7 – Performance headline, persistent messages, and server channels

Note: the large figures in the table above show the peak number of round trips per second—within the range of the test, and the number of driving applications used to achieve the throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

2.3.3 'runmqlsr' vs. inetd 'amqcrsta' listener – server channels

For the following distributed queuing measurements, the messaging rate used is 1 round trip per driving application per *second*, i.e. a request message outbound over the sender channel, and a reply message inbound over the receiver channel per second. Note that there are a fixed number of 4 server channel pairs for the nonpersistent messaging tests, and 2 pairs for the persistent message tests. These tests also compare the difference between *nonthreaded* channels (the 'amqcrsta' process started by inetd, and the 'runmqchl' process started by the queue manager) with *threaded* channels (the 'runmqlsr' process started by the user, and the 'runmqchi' process started with the queue manager).

Figure 14 shows that there is little difference between the inetd 'amqcrsta' and 'runmqlsr' listener in terms of the number of round trips that can be achieved per second before the round trip response time exceeds one second. However, there is greatly improved scalability at more then 1250 apps.

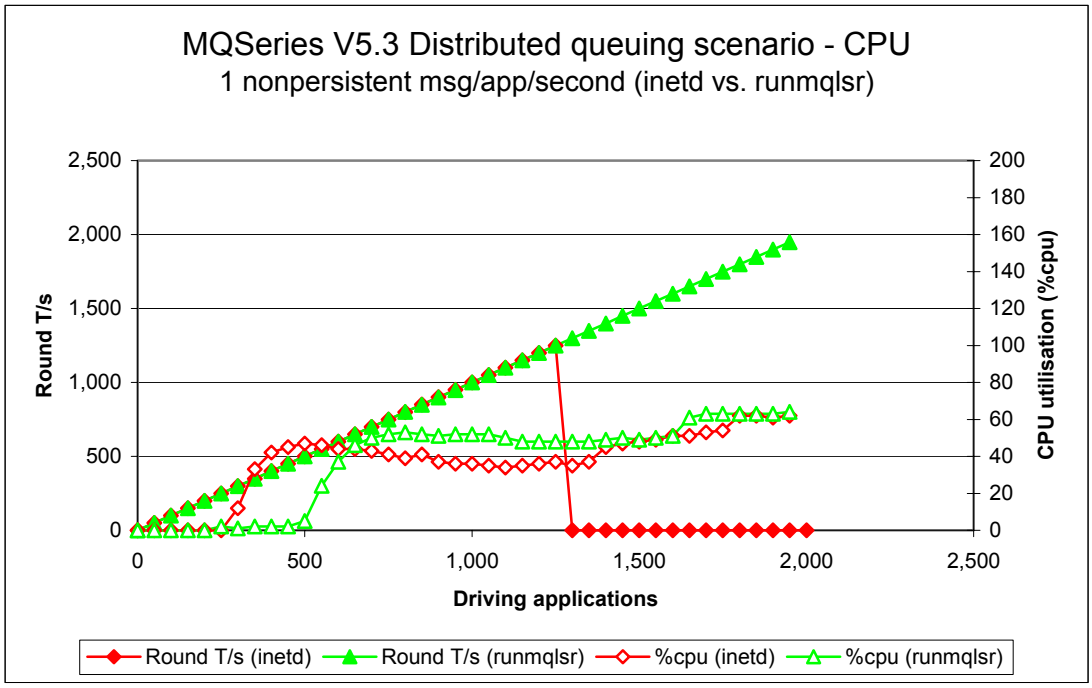

Figure 14 – 'runmqlsr' vs. inetd 'amqcrsta' listener, server channels

Note: messaging in these tests is 1 round trip per driving application per *second*.

Figure 15 below show that there is little difference between the inetd 'amqcrsta' and 'runmqlsr' listener in terms of the number of round trips that can be achieved per second before 250 apps. However, there is greatly improved scalability at more then 250 apps.

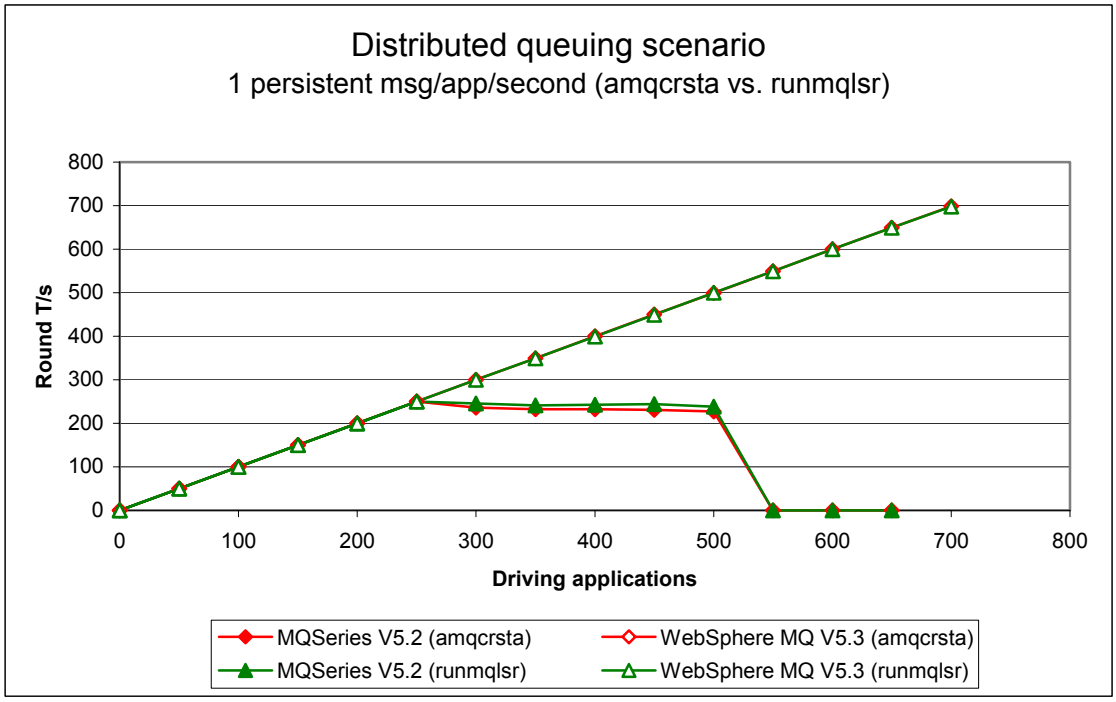

Figure 15 – 'runmqlsr' vs. inetd 'amqcrsta' listener, server channels

Test name	Apps	Rate/App/hr	Round T/s	%	Resp time (s)
Dqnp1_r3600 (inetd) (MQSeries V5.2)	1,250 (1,000)	3,600	1,249 (999)	+25	0.006 (0.005)
Dqnp1_r3600_runmqlsr (MQSeries V5.2)	3,150 (2,800)	3,600	3,147 (2,798)	+12	0.022 (0.010)
dqpm1_r3600 (inetd) (MQSeries V5.2)	500 (250)	3,600	500 (250)	+100	0.232 (0.266)
dqpm1_r3600_runmqlsr (MQSeries V5.2)	750 (250)	3,600	725 (250)	+190	0.384 (0.263)

Table 8 – 1 round trip per driving application per *second*, server channels

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

3 Large messages

3.1 MQI response times: 50bytes to 2MB – local queue manager

Figure 16 and **Figure 17** below show that the response for MQPUT/GET pairs is improved for persistent message sizes from 5K to 2MB.

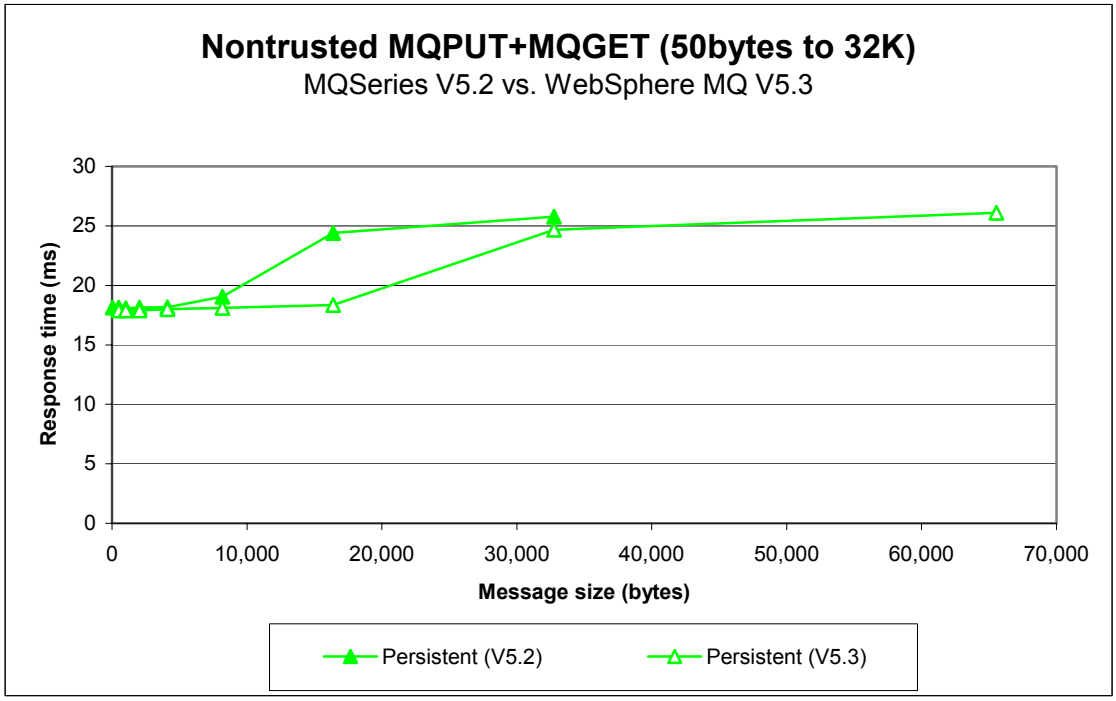

Figure 16 – Effect of message size on MQI response time (50byte - 32K)

Note: messaging in these tests is with no think-time.

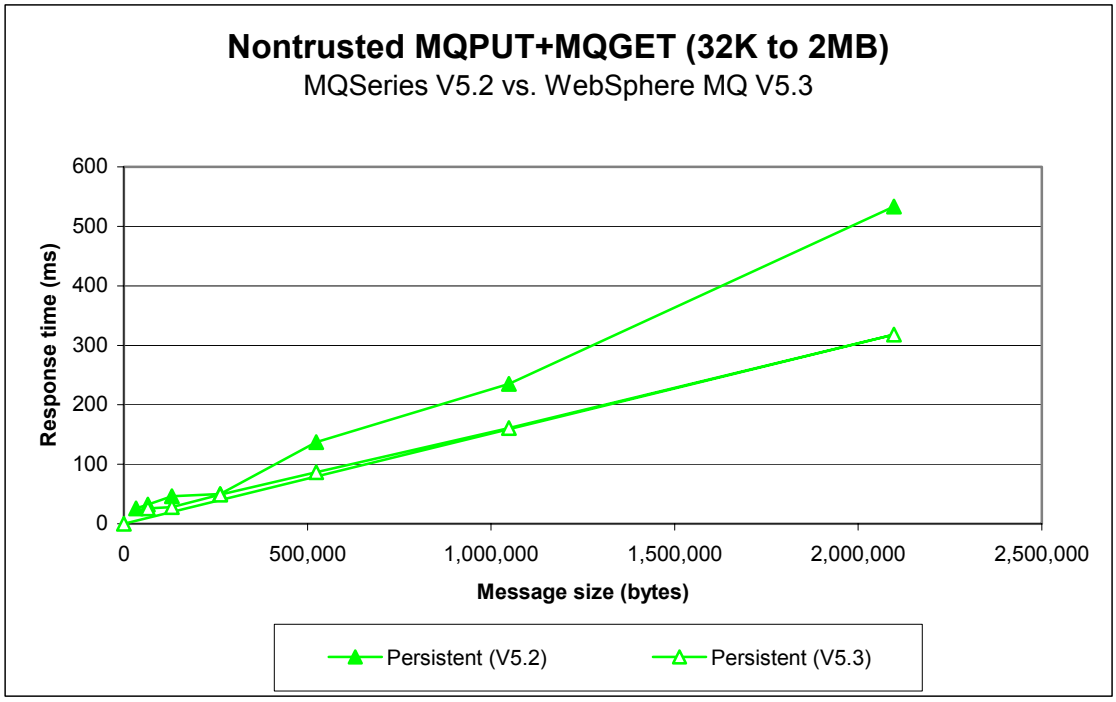

Figure 17 – Effect of message size on MQI response time (32K - 2MB)

Note: messaging in these tests is with no think-time.

Figure 18 and **Figure 19** show that the response for MQPUT/GET pairs is improved for persistent message sizes from 250K to 2MB

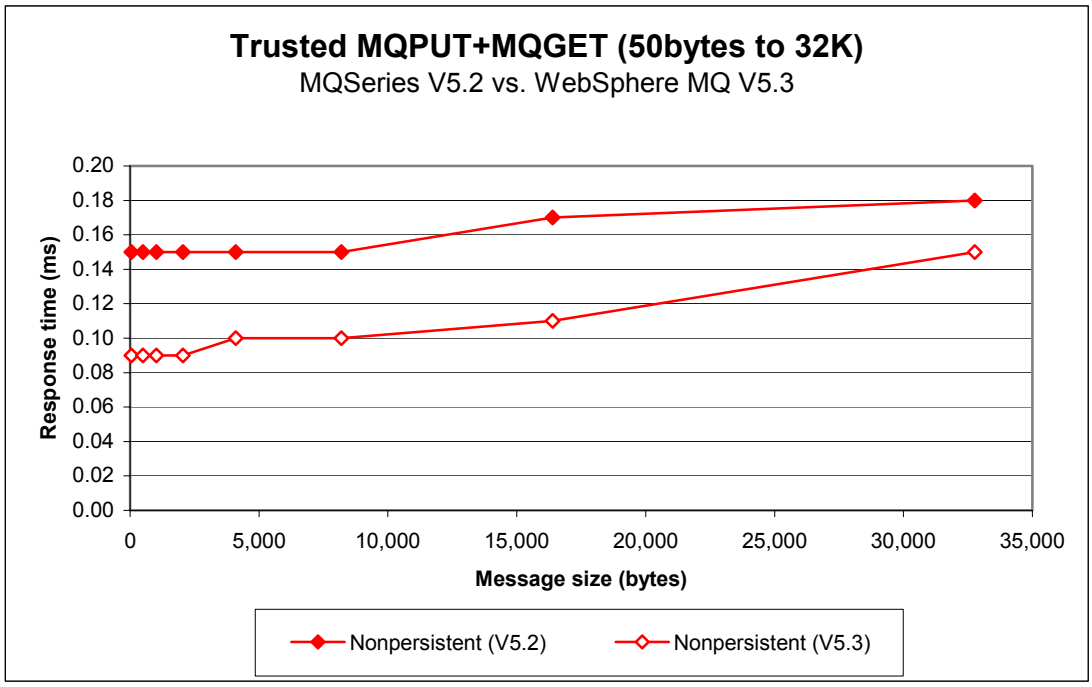

Figure 18 – Effect of message size on trusted MQI response time (50byte - 32K)

Note: messaging in these tests is with no think-time.

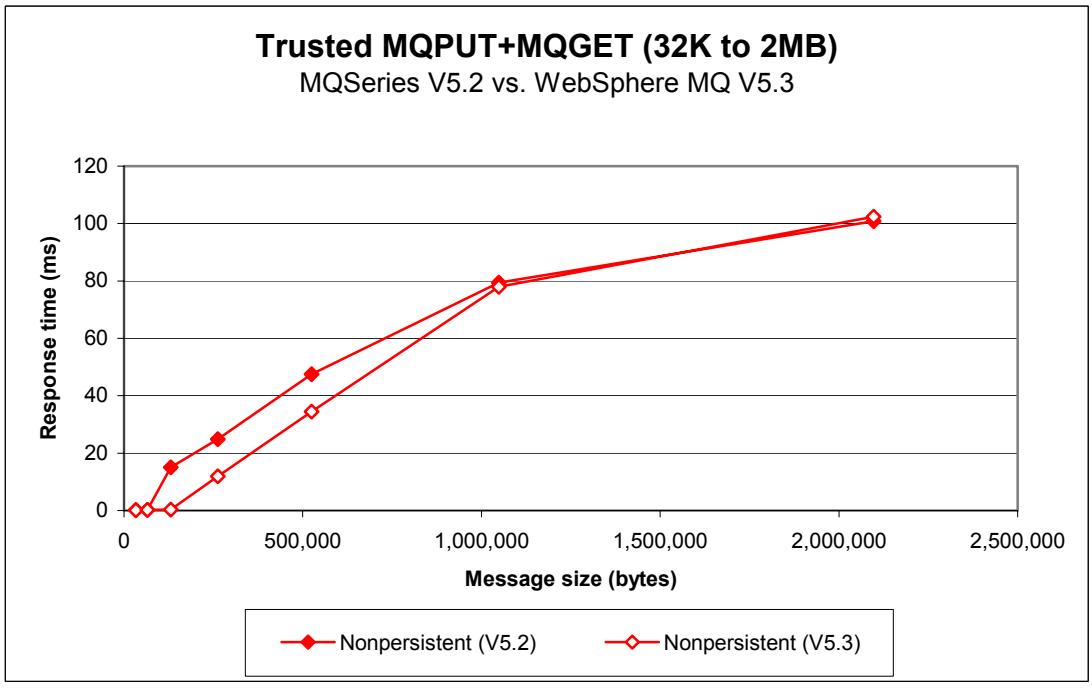

Figure 19 – Effect of message size on trusted MQI response time (32K - 2MB)

Note: messaging in these tests is with no think-time.

3.2 Large messages: 20K and 200K – local queue manager

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
local_np1_2K (MQSeries V5.2)	8 (5)	2K	5,392 (3,167)	+70	0.002 (0.002)
local_np1_20K (MQSeries V5.2)	5 (4)	20K	2,816 (1,944)	+45	0.002 (0.002)
local_np1_200K (MQSeries V5.2)	2 (3)	200K	311 (247)	+26	0.007 (0.013)
local_pm3_2K (MQSeries V5.2)	120 (20)	2K	781 (247)	+216	0.184 (0.090)
local_pm3_20K (MQSeries V5.2)	60 (28)	20K	212 (166)	+28	0.439 (0.191)
local_pm3_200K (MQSeries V5.2)	16 (20)	200K	31 (31)	0	0.587 (0.756)

Table 9 – 2K, 20K and 200K messages, local queue manager

Note: messaging in these tests is with no think-time.

Note: the figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

The measurements for 200K *persistent* messages show that there is little difference in the performance of large messages between Version 5.2 and Version 5.3—this is because most of the time taken by the queue manager is in logging the messages to disk.

Figure 20 below shows how the throughput of small nonpersistent messages is improved in WebSphere MQ V5.3, using any number of driving applications. For 2k applications up to 70% peak throughput, and up to 45% peak throughput for 20k messages

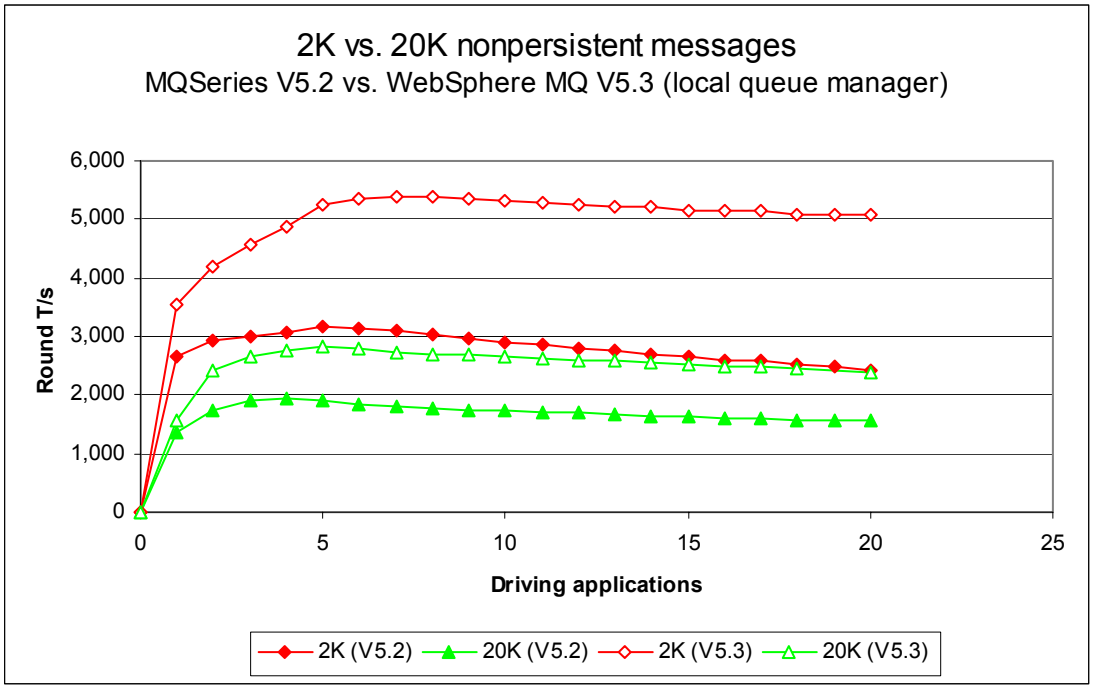

Figure 20 – 2K and 20K nonpersistent messages, local queue manager

Figure 21 below shows how the throughput of 2K persistent messages is improved significantly (216% peak throughput in round trips per second), and the throughput of 20K persistent messages is improved slightly, in WebSphere MQ V5.3, using more than 8 driving applications.

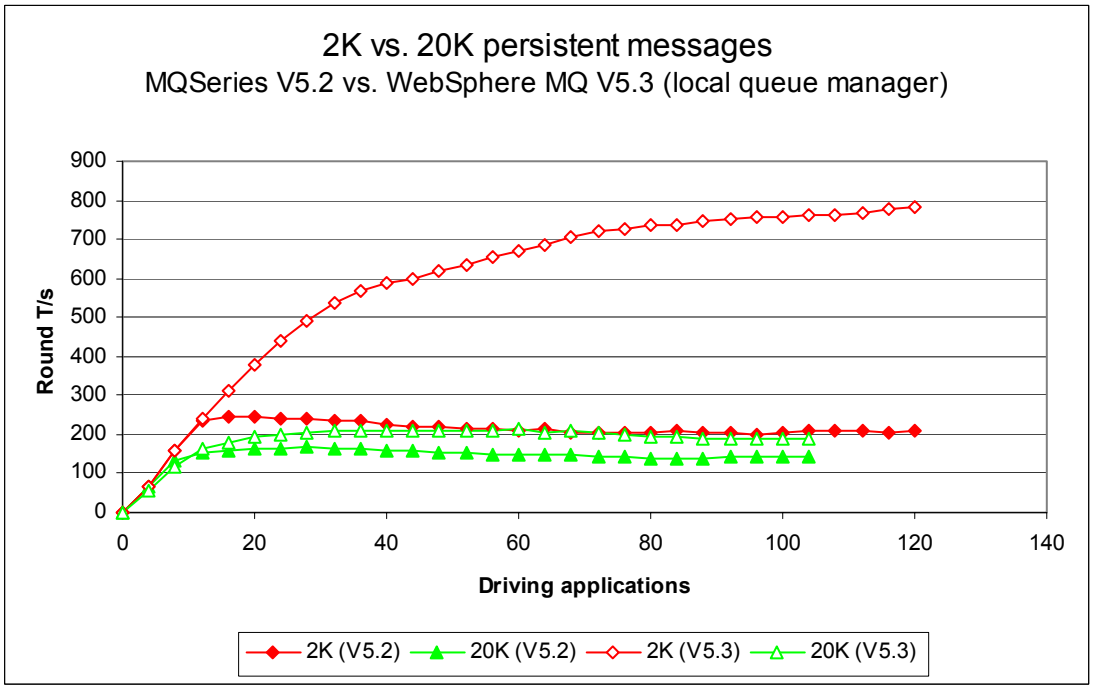

Figure 21 – 2K and 20K persistent messages, local queue manager

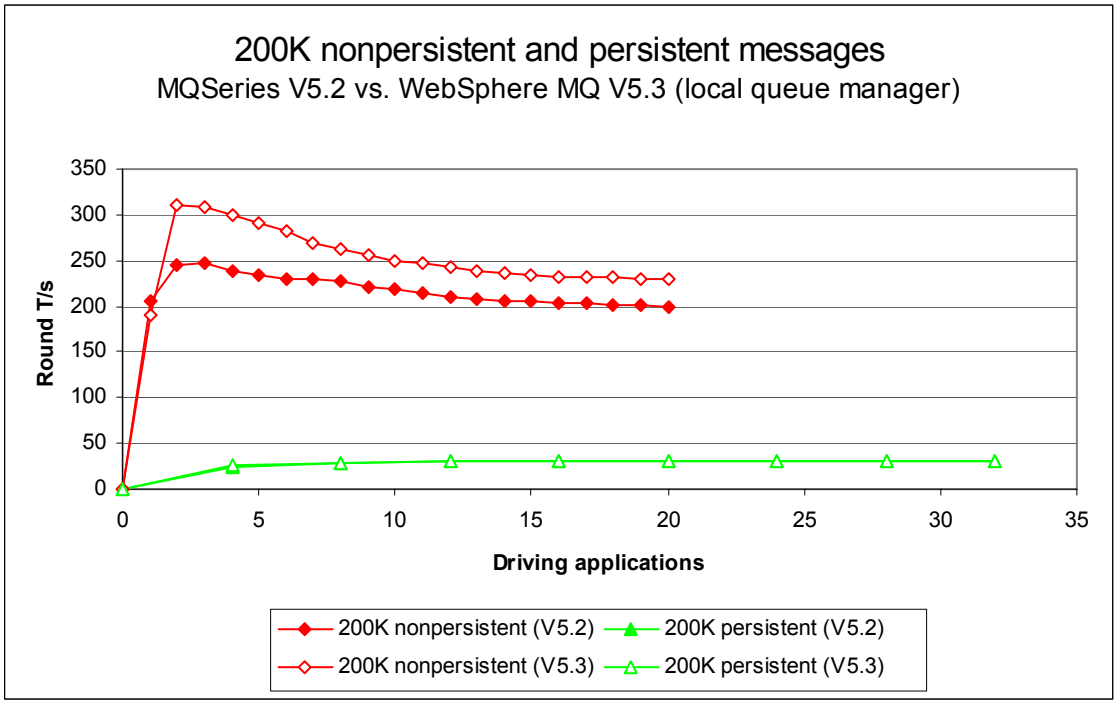

Figure 22 – 200K nonpersistent and persistent messages, local queue manager

*Note in the above graph the 200k persistent (V 5.2) line is behind the 200k (V 5.3) line.

Large messages: 20K and 200K – client channels

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
clnp1 (MQSeries V5.2)	11 (7)	2K	3,913 (2,660)	+47	0.003 (0.003)
clnp1_20K (MQSeries V5.2)	12 (11)	20K	1,181 (1,045)	+13	0.012 (0.012)
clnp1_200K (MQSeries V5.2)	6 (6)	200K	110 (108)	+2	0.064 (0.064)
clpm3 (MQSeries V5.2)	120 (16)	2K	711 (239)	+197	0.203 (0.076)
clpm3_20K (MQSeries V5.2)	60 (20)	20K	212 (157)	+35	0.439 (0.150)
clpm3_200K (MQSeries V5.2)	24 (28)	200K	28 (28)	0	0.989 (0.978)

Table 10 – 2K, 20K and 200K messages, client channels

Note: messaging in these tests is with no think-time, and the inetd 'amqcrsta' listener.

Note: the figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

The measurements for 200K *persistent* messages show that there is little difference in the performance of large messages between Version 5.2 and Version 5.3—this is because most of the time taken by the queue manager is in logging the messages to disk.

Figure 23 below shows how the throughput of small (2K) nonpersistent messages is improved by 47% peak throughput in round trips per second in WebSphere MQ V5.3, using 5 or more driving applications.

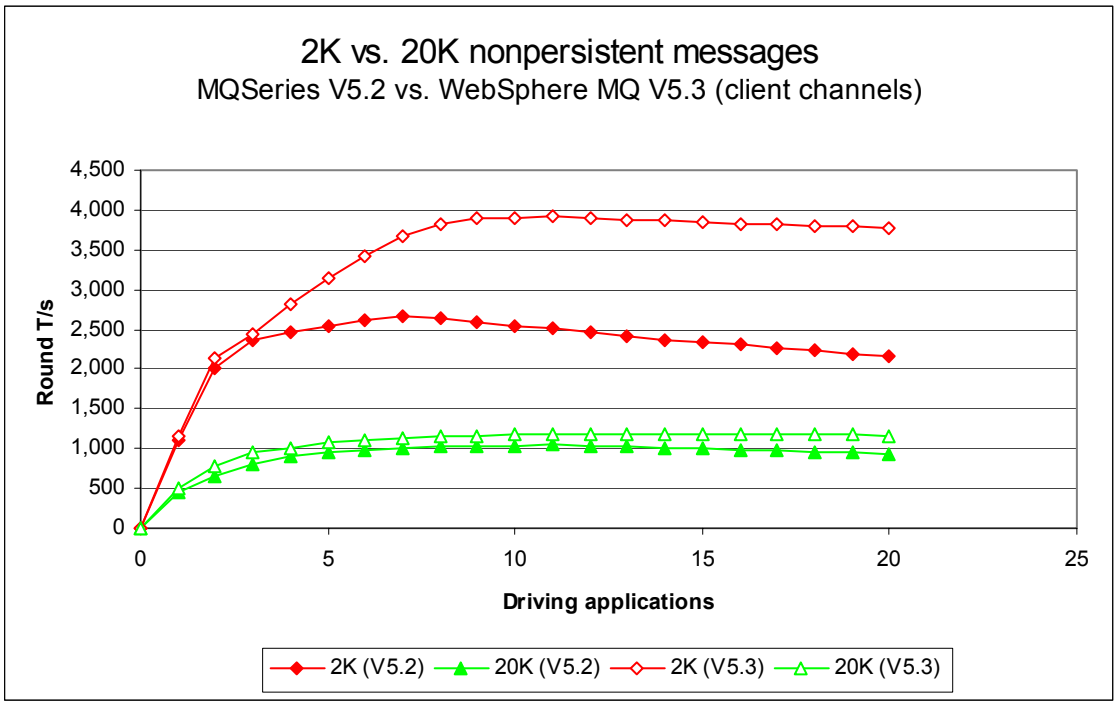

Figure 23 – 2K and 20K nonpersistent messages, client channels

Figure 24 below shows how the throughput of 2K persistent messages is improved significantly, and the throughput of 20K persistent messages is improved slightly, in WebSphere MQ V5.3, using more than 16 driving applications. Using 80 or more driving applications Version 5.3 gives a *maintained* persistent throughput of 700 round trips or more per second (71% more than Version 5.2).

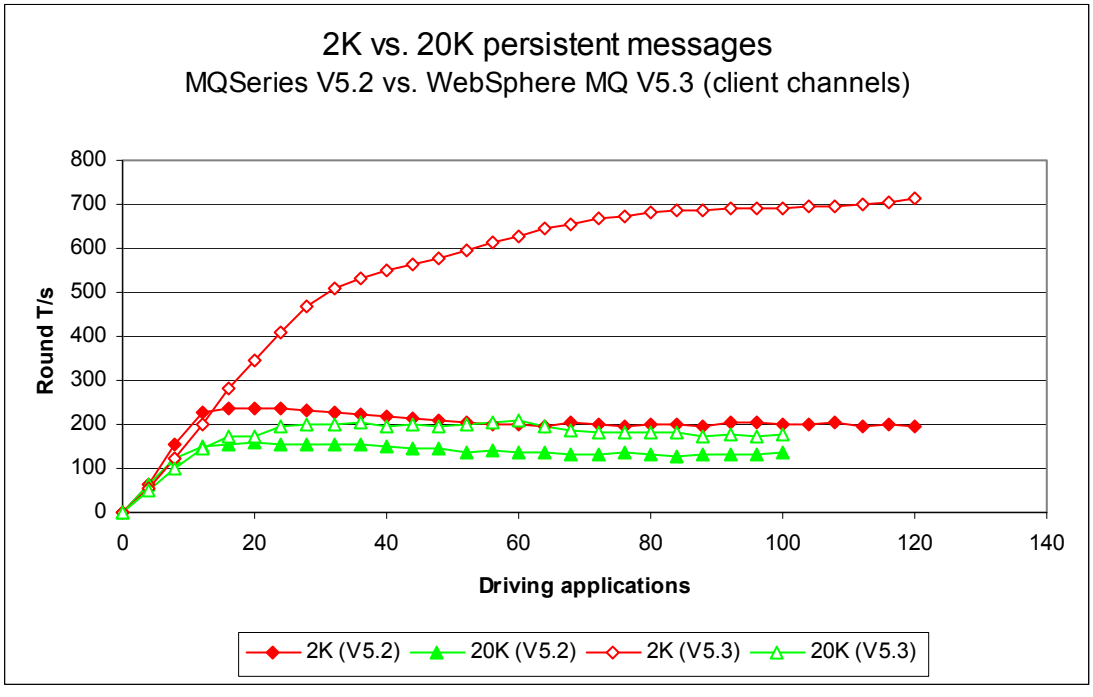

Figure 24 – 2K and 20K persistent messages, client channels

Figure 25 below shows that there is almost no difference in performance in 200k messages over MQI client channels.

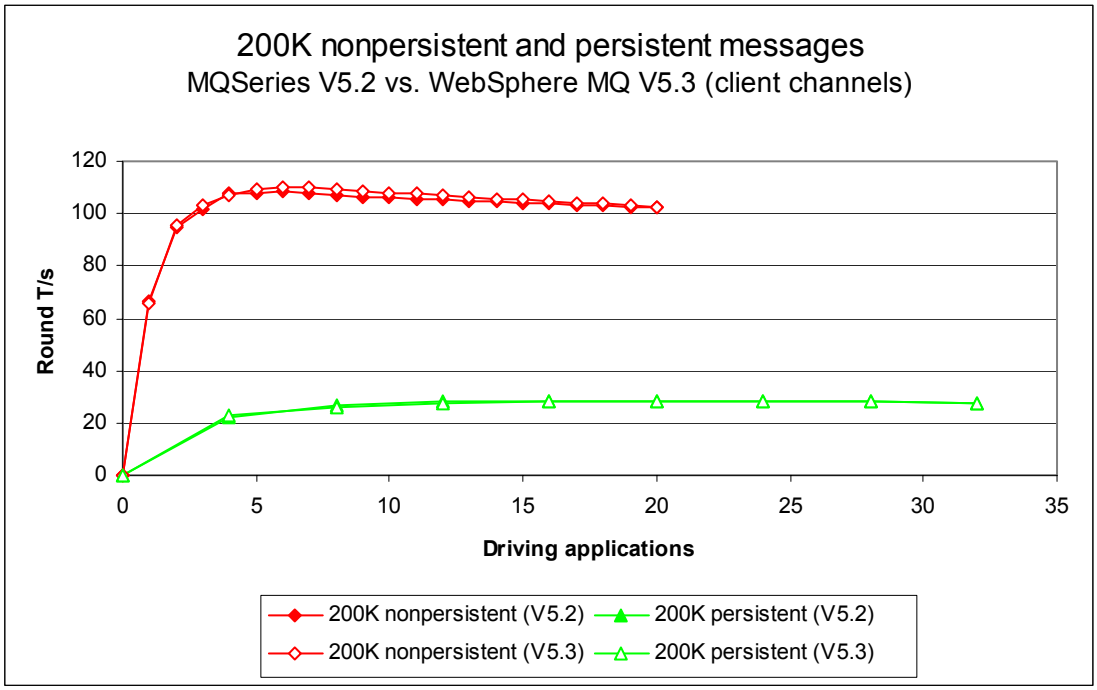

Figure 25 – 200K nonpersistent and persistent messages, client channels

*Note in the above graph the 200k persistent (V 5.2) line is behind the 200k (V 5.3) line.

3.3 Large messages: 20K and 200K – distributed queuing

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
dqnp1 (MQSeries V5.2)	19 (16)	2K	2,128 (2,021)	+5	0.010 (0.009)
dqnp1_20K (MQSeries V5.2)	7 (10)	20K	1,076 (985)	+9	0.007 (0.012)
dqnp1_200K (MQSeries V5.2)	16 (11)	200K	83 (29)	+186	0.231 (0.440)
dqpm1 (MQSeries V5.2)	236 (108)	2K	459 (353)	+30	0.627 (0.365)
dqpm1_20K (MQSeries V5.2)	68 (56)	20K	79 (78)	+1	0.992 (0.965)
dqpm1_200K (MQSeries V5.2)	8 (4)	200K	10 (7)	+43	0.888 (0.583)

Table 11 – 2K, 20K and 200K messages, server channels

Note: messaging in these tests is with no think-time, and the inetd ‘amqcrsta’ listener.

Note: the figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

The measurements for 20K and 200K *persistent* messages show that there is little difference in the performance of large messages between Version 5.2 and Version 5.3—this is because most of the time taken by the queue manager is in logging the messages to disk.

Figure 26 below shows how the throughput of small (2K) nonpersistent messages is improved in WebSphere MQ V5.3, using as few as 3 driving applications.

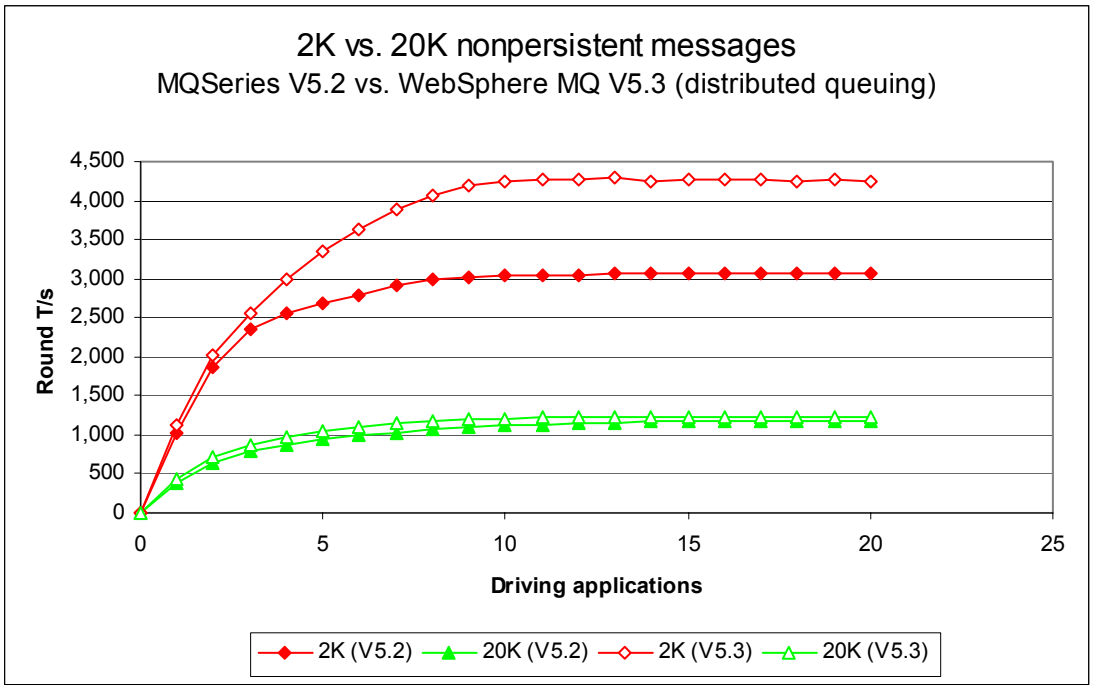

Figure 26 – 2K and 20K nonpersistent messages, server channels

Figure 27 below shows how the throughput of 2K persistent messages is improved significantly, and the throughput of 20K persistent messages is improved slightly, in Version 5.3 using more than 40 driving applications. Using more than 40 driving applications Version 5.3 gives better throughput (compared to Version 5.2) and V.5.3 shows greater scalability.

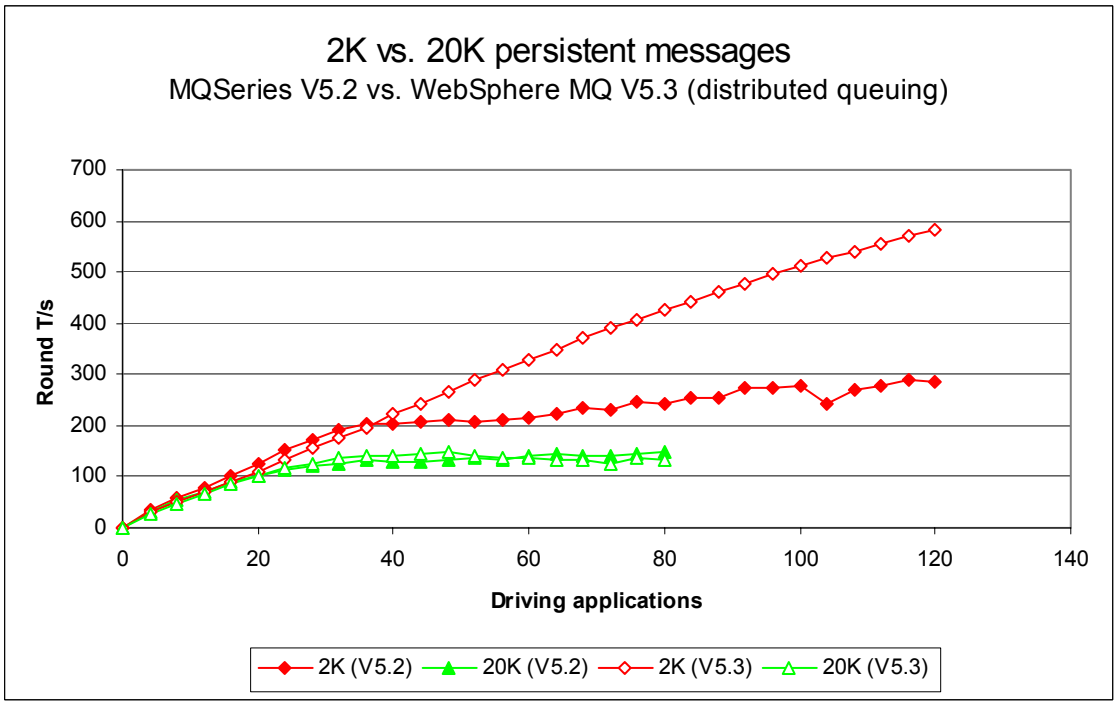

Figure 27 – 2K and 20K persistent messages, server channels

The 200K nonpersistent message tests were intentionally designed to finish at 20 driving applications. Using more than 10 driving applications, message throughput levels out. The 200K persistent message tests were designed to finish at 120 driving application, but after 28 driving applications (both Version 5.2 and Version 5.3) the tests approach the response time criteria.

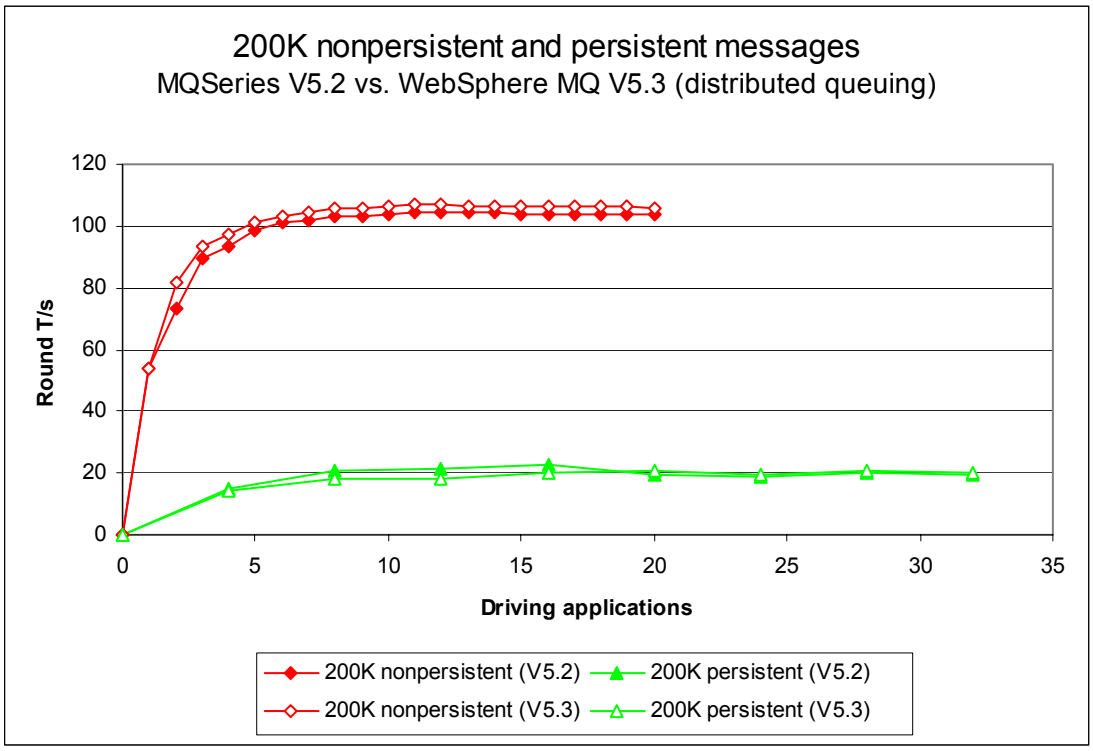

Figure 28 – 200K nonpersistent and persistent messages, server channels

4 Trusted server application

Figure 29 and **Table 12** show that peak persistent message throughput has increased by 228% (267 cf. 875 RT/s) comparing Version 5.2 to Version 5.3, also with the advantage of improved scalability after 3 driving applications. In addition, peak nonpersistent message throughput has increased by 50% (5,130 cf. 7,678 RT/s)

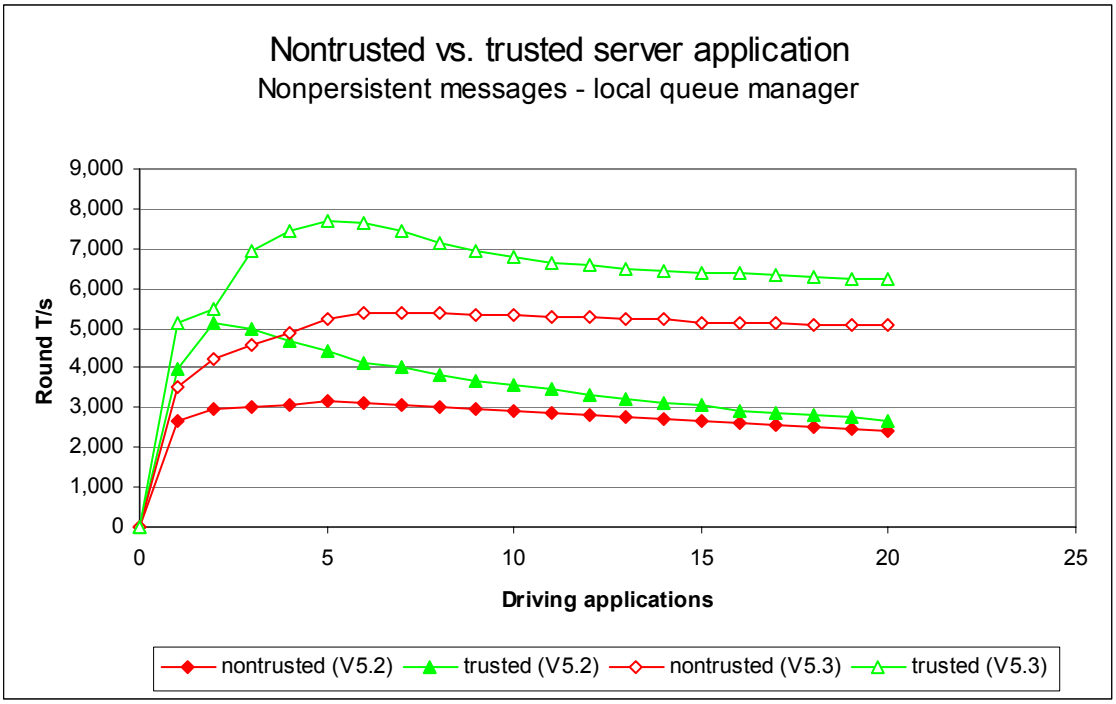

Figure 29 – Trusted server application, local queue manager

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
local_np1_trusted (MQSeries V5.2)	7 (2)	2K	7,678 (5,130)	+50	0.001 (0.001)
local_pm1_trusted (MQSeries V5.2)	120 (16)	2K	875 (267)	+228	0.171 (0.070)

Table 12 – Trusted server application, local queue manager

Note: messaging in these tests is with no think-time.

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

5 MQSeries Tuning recommendations

5.1 Tuning the queue manager

This section highlights the tuning activities that are known to give performance benefits for WebSphere MQ V5.3; all of these can be applied equally to Version 5.2. The reader should note that the following tuning recommendations **might not** necessarily **need** to be applied, especially if the message throughput and/or response time of the queue manager system already meets the required level. Some tuning recommendations that follow may degrade the performance of a previously balanced system if applied inappropriately. The reader should carefully monitor the result of tuning the queue manager to be satisfied that there have been no adverse effects.

Customers should test that any changes have not used excessive real resources in their environment and make only essential changes. For example, allocating several megabytes for multiple queues reduces the amount of shared and virtual memory available for other subsystems, as well as over committing real storage

5.1.1 Queue disk, Log disk, and message persistence

To avoid potential queue and log I/O contention due to the queue manager simultaneously updating a queue file and log extent on the same disk, it is important that queues and logs are located on *separate* and *dedicated* physical devices. With the queue and log disks configured in this manner, careful consideration must still be given to message persistence: persistent messages should only be used if the message needs to survive a queue manager restart (forced by the administrator or as the result of a power failure, communications failure, or hardware failure). In guaranteeing the recoverability of persistent messages, the path length through the queue manager is three times longer than for a nonpersistent message. This overhead does not include the additional time for the message to be written to the log, although this can be minimised by using cached disks.

5.1.1.1 Nonpersistent queue buffer

The default nonpersistent queue buffer size is 64K per queue. This can be increased to 1MB using the TuningParameters stanza and the *DefaultQBufferSize* parameter. The nonpersistent queue buffer is computationally less expensive because the queue manager does not have to retrieve the message from the queue file. Increasing the queue buffer provides the capability to absorb peaks in message throughput at the expense of real storage, but it is **not** suitable as a long-term storage for nonpersistent messages as this buffer is **not** recovered after a queue manager restart. Defining queues using large nonpersistent queue buffers can degrade performance either if the system is short of real memory because a large number of queues have already been defined with large buffers, or for other reasons—e.g. a large numbers of channels defined.

Note: the nonpersistent queue buffer is allocated in shared storage so consideration must be given to whether the agent process or application process has the memory addressability for all the required shared memory segments.

Queues can be defined with different values of *DefaultQBufferSize.* If some queues need to be defined differently to others, the values can be set in the TuningParameters stanza. When the queue manager is restarted, existing queues will keep their earlier definitions and new queues will be created with the desired parameters. When a queue is opened, resources are allocated according to the definition held on disk from when the queue was created.

5.1.2 Log buffer size, Log file size, and number of log extents

To improve persistent message throughput the *LogBufferPages* should be increased to its maximum configurable size of *512* x 4K pages = 2MB, the *LogFilePages* (i.e. `crtmqm -lf <LogFilePages>`) should be configured to a large size, for example: *16384* x 4K pages = 64MB, and the number of *LogPrimaryFiles* (i.e. `crtmqm -lp <LogPrimaryFiles>`) should be configured to a large number. The cumulative effect of this tuning will:

- improve the throughput of persistent messages (permitting a possible maximum 2MB of log records to be written from the log buffer to the log disk in a single write),
- reduce the frequency of log switching (permitting a greater amount of log data to be written into one extent),
- Allow more time to prepare new linear logs or recycle old circular logs (especially important for long-running units of work).

Changes to the queue manager LogBufferPages parameter takes effect at the next queue manager restart. The number of pages can be changed for all subsequent queue managers by changing the LogBufferPages parameter in the product default Log stanza.

It is unlikely that poor persistent message throughput will be attributed to the 2MB limit of the queue manager log buffer. It is possible to fill and empty the log buffer several times each second and reach a CPU limit writing data into the log buffer, before a log disk bandwidth limit is reached.

5.1.3 Channels: process or *thread*, standard or *fastpath*?

It is no longer necessary to consider the system design when deciding whether it is preferable to configure inetd to use process channels (‘amqcrsta’, and for server channels an MCATYPE of ‘PROCESS’), or use threaded channels (‘runmqlsr’, and for server channels an MCATYPE of ‘THREAD’). Prior to Version 5.3, it was necessary to use more than one ‘runmqlsr’ listener using more than one port. ‘runmqlsr’ **can** now **be used** in **all** scenarios with client and server channels. Additional resource savings are available using ‘runmqlsr’, including a reduced requirement on: virtual memory, number of processes file handles (***nfile***), and System V IPC (***shmmax***, ***semmni***, and ***shmseg***).

Fastpath channels, and/or fastpath applications—see later paragraph for further discussion, can increase throughput for both nonpersistent and persistent messaging. For persistent messages, the improvement is only for the path through the queue manager, and does not affect performance writing to the log disk. The reader should note that since the greater proportion of time for persistent messages is in the queue manager writing to the log disk, the performance improvement for fastpath channels is less apparent with persistent messages than with nonpersistent messages.

5.2 Application design and configuration

5.2.1 *Standard* or fastpath?

The reader should be aware of the issues associated with writing and using fastpath applications—described in the 'MQSeries Application Programming Guide'. Although it **is recommended** that customers use fastpath channels, it is **not recommended** to use fastpath applications. If the performance gain offered by running fastpath is not achievable by other means, it is essential that applications are rigorously tested running fastpath, and never forcibly terminated (i.e. the application should always disconnect from the queue manager). Fastpath channels are documented in the 'MQSeries Intercommunication Guide'.

5.2.2 Parallelism, batching, and triggering

An application should be designed wherever possible to have the capability to run *multiple instances* or *multiple threads* of execution. Although the capacity of a multi-processor system can be fully utilised with a small number of applications using nonpersistent messages, more applications are required if the workload is mainly using persistent messages. Processing messages inside syncpoint can help reduce the amount of time the queue managers takes to write a batch of persistent messages to the log disk. The performance profile of a workload will also be subject to variability through cycles of low and high message volumes, therefore a degree of experimentation will be required to determine an optimum configuration.

Queue avoidance is a feature of the queue manager that allows messages to be passed directly from an 'MQPUT'er to an 'MQGET'er without the message being placed on a queue. This feature only applies for processing nonpersistent messages outside of syncpoint. In addition to improving the performance of a workload with multiple parallel applications, the design should attempt to ensure that an application or application thread is always available to process messages on a queue (i.e. an 'MQGET'er). Then nonpersistent messages outside of syncpoint do not need to be *physically* placed on a queue.

The reader should note that as more applications are processing messages on a single queue there is an increasing likelihood that queue avoidance will not be maintainable. The reasons for this have a cumulative and exponential effect, for example, when nonpersistent messages are being placed on a queue quicker than they can be removed. The first effect is that messages begin to fill the nonpersistent queue buffer—and MQGETers need to retrieve messages from the buffer rather than being received directly from an MQPUTer. A secondary effect is that as messages are spilled from the buffer to the queue disk, the MQGETers must wait for the queue manager to retrieve the message from the queue disk rather than being retrieved from the queue buffer. While these problems can be addressed by configuring for more MQGETers (i.e. processing threads in the server application), or using a larger nonpersistent queue buffer, it may not be possible to avoid a performance degradation.

Processing messages inside syncpoint, (i.e. in batches) can be more efficient than outside of syncpoint. As the number of messages in the batch increases, the average processing cost of each message decreases. For persistent messages the queue manager can write the entire batch of messages to the log disk in one go—outside of syncpoint control, the queue manager must wait for each message to be written to the log before return control to the application.

The 'runmqlsr' has a much smaller overhead of connecting to and disconnecting from the queue manager because it does not have to create a new process. Furthermore, in Version 5.3 the maximum number of connections into a single 'runmqlsr' listener has been significantly increased making it the preferred method of running short sessions over client channels. Nevertheless, the implementation of triggering is still worth consideration with regard to programming a disconnect interval as an input parameter to the application. This can provide the flexibility to make tuning adjustments in a production environment, if for instance, it is more efficient to remain connected to the queue manager between periods of message processing, or disconnect to free queue manager and Operating System resources.

Appendix A Measurement environment

A.1 Workload description

A.1.1 MQI response time tool

The MQI tool exercises the local queue manager by measuring elapsed times of the 8 main MQSeries verbs: MQCONN(X), MQDISC, MQOPEN, MQCLOSE, MQPUT, MQGET, MQCMIT, and MQBACK. The following MQI calls are paired together inside a test application:

- MQCONN(X) and MQDISC,
- MQOPEN and MQCLOSE,
- MQPUT and MQGET,
- MQCMIT and MQBACK with MQPUT and MQGET.

Note: MQCLOSE elapsed time is only measured for an empty queue.

Note: performance of MQCMIT and MQBACK is measured in conjunction with MQPUT and MQGET, putting and getting messages inside a unit of work (i.e. inside syncpoint control). The unit of work is committed at the end of each batch. The number of messages per batch is a parameter of the test.

Note: the performance of verbs: MQSET, MQINQ, or MQBEGIN are not measured.

A1.2 Test scenarios workload

A1.2.1 The driving application programs

The test scenario workload simulates many driving applications running on a single driving machine. This is not typical of a customer environment and is only used to facilitate test coordination. Driving applications were multi-threaded with each thread performing a sequence of MQI calls. The number of threads in each application was adjusted according to whether the test was measuring a local queue manager, a client channel, or distributed queuing scenario. This was done to reduce storage overheads on the driving system. Each driving application thread performed the sequence of actions as outlined in the test scenario illustrations in the '**Performance headlines**' starting on **page 4**.

Message size: For the release highlights and performance headlines (including rated messaging tests), a 2K message size was used. For the large message measurements a 20K and 200K message size was used.

Message rate: In all but the *rated* and *capacity limit* tests, message processing was performed in a *tight-loop*. In the *rated* tests, a message rate of 1 round trip per driving application per *second* was used, and in the *capacity limit* tests a message rate of 1 round trip per channel per *minute* was used.

Nonpersistent and persistent messages were used in tests.

Note: the driving applications gathered timing information for all MQI calls using a high-resolution timer.

A1.2.2 The server application program

The server application is written as a multi-threaded program configured to use 5 threads for processing nonpersistent messages, and 20 or more threads to process persistent messages. Each server thread performed the sequence of actions as outlined in the test scenario illustrations in the '**Performance headlines**' starting on **page 4**.

Nonpersistent messaging is done outside of syncpoint control. Persistent messaging is done inside of syncpoint control. The average message throughput expressed as a number of round trips per second was calculated and reported by the server program.

A1.3 Test Descriptions

Local_np: A local test with both driver and server applications running on the same hardware. The test uses non-persistent messages and unless otherwise stated tests are designed to run to 20 client connections. The aim of this test is to get the highest possible throughput without network or MCA restrictions.

Local_pm: A local test with both driver and server applications running on the same hardware. The test uses persistent messages and unless otherwise stated tests are designed to run to 120 client connections. The aim of this test is to get the highest possible throughput without network or MCA restrictions.

Queue manager log configuration:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

Clnp A test where clients connect using MQI-client channels to a remote queue manager on separate physical hardware, the server is still local to the queue manager. The test uses non-persistent messages and unless otherwise stated tests are designed to run to 20 client connections. The aim is to provide measurements for client applications (a common set-up for customers).

Clpm A test where clients connect using MQI-client channels to a remote queue manager on separate physical hardware, the server is still local to the queue manager. The test uses persistent messages and unless otherwise stated tests are designed to run to 120 client connections. The aim is to provide measurements for client applications (a common set-up for customers).

Queue manager log configuration:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

Dqnp A distributed test, the clients connect to a local queue manager and the server to another local queue on different hardware, the two queue managers then communicate via server channels. The test uses non-persistent messages and unless otherwise stated tests are designed to run to 20 client connections. The aim of this test is to provide measurement of the server channels and comparison to client and local tests.

Dqpm A local distributed test, the clients connect to a local queue manager and the server to another local queue on different hardware, the two queue managers then communicate via server channels. The test uses persistent messages and unless otherwise stated tests are designed to run to 120 client connections. The aim of this test is to provide measurement of the server channels and comparison to client and local tests.

Queue manager log configuration:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

MQPUT + MQGET	For this test a simple application records the time taken to do the MQ verbs MQPUT and MQGET, each verb is repeated 5000 times and the 90% percentile is shown in the graph. The tests are repeated for both persistent and non-persistent messages. Also, run both trusted and nontrusted. Queue manager log configuration: `LogPrimaryFiles=3, LogFilePages=2048`
Trusted server	The trusted tests are run with the same settings and scenarios as the other tests however in the trusted tests the server application connects directly to the queue manager rather then through an agent process. This should give notable performance benefits, however it is not recommended for general usage.

A.2 Hardware

Unless stated otherwise one or two machines of the following specification were used for all of the tests in this report

Model:	IBM Netfinity 8500R
Processor:	Intel Pentium 3 Xeon 700Mhz, 2 MB L2 cache
Architecture:	4-way SMP
Memory (RAM):	8GB
Disk:	2 Internal 10,000 rpm SCSI disks 18GB and 9GB
	1 External 10,000 rpm SCSI disks – 9GB
Network:	1GBit Ethernet

A.3 Software

Linux O/S:	Red Hat v 7.3 (2.4.18 kernel)
MQSeries:	Version 5.3 (B.11.530.00), and Version 5.2 (B.11.520.00)
Compiler:	Linux POSIX-conforming C compiler

Glossary

Test name	The name of the test Note: the test names in some cases are rather long. This is done to provide a descriptive qualification of the test measurement to relate to the performance discussion in the sections throughout the document: **local** => local queue manager test scenario **cl** => client channel test scenario **dq** => distributed queuing test scenario **np** => nonpersistent messages **pm** => persistent messages **r3600** => 1 round trip per driving application per second **runmqlsr** => channels using the 'runmqlsr' listener (client channel test scenario, in addition to 'runmqchi' for distributed queuing test scenarios) **c6000** => 6,000 client driving applications (i.e. 6,000 MQI-client connections) **q1000** => 1,000 server channel pairs **max** => maximum number of channels (or channel pairs) **no_correl_id** => correlation identifier not used in the response messages (as each response is placed on a unique reply-to queue per driving application)
Apps	The number of driving applications connected to the queue manager at the point where the performance measurement is given
Rate/App/hr	The target message throughput rate of each driving application
Round T/s	The average achieved message throughput rate of all the driving applications together, measured by the server application
% (Round T/s)	The percentage increase in the total message throughput rate Note: the nature of the comparison is noted under each table where percentage improvements have been given
Resp time (s)	The average response time each round trip, as measured and averaged by all the driving applications
CURDEPTH	The number of messages on the input queue as a snapshot Note: runmqsc <qmname>, DISPLAY QLOCAL(<qname>) CURDEPTH
queue disk (kbps)	The queue disk kilobytes transferred per second
Swap	The total amount of swap area reservation for all processes in MB, unless otherwise specified as swap/app (i.e. swap area reservation per driving application) Note: swap area is reserved for ALL allocated virtual memory whether the process needs it, is physically using it, or not. This is enforced by the HP-UX kernel to ensure a process can use ALL its allocated swap should the need arise
Shm	The amount of allocated System V IPC shared memory in MB
Segs	The number of System V IPC shared memory segments
sems	The number of System V IPC semaphores

*** end of document ***